Microsoft

SQL Server Transactions and Concurrency

Module 3

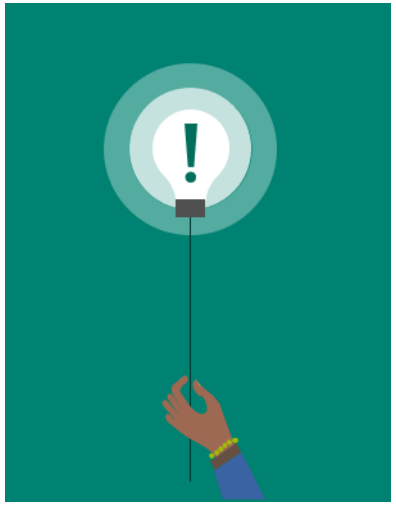
Learning Units covered in this Module

- Lesson 1: SQL Server Transactions
- Lesson 2: SQL Server Isolation Levels
- Lesson 3: SQL Server Locking
- Lesson 4: Troubleshooting Concurrency Performance

Objectives

After completing this learning, you will be able to:

- Understand what is a transaction and its types.
- Recommendations for using transactions.
- Monitor current transactions in SQL Server.
- Describe the concurrency and concurrency types.
- Blocking and Deadlocking



Lesson 1: SQL Server Concurrency and Transactions

What is a Transaction?

A transaction is a sequence of steps that perform a logical unit of work.

Must Exhibit ACID properties, to qualify as a transaction.

A - Atomicity

- A transaction is either fully completed or not at all.

C - Consistency

- A transaction must leave data in a consistent state.

I - Isolation

- Changes made by a transaction must be isolated from other concurrent transactions.

D - Durability

- The modifications persist even in the event of a system failure.

Transaction Modes

Auto-Commit

Individual statements that complete successfully, will be committed. If errors are encountered the statement is rolled back.

Explicit

Transaction is explicitly defined with a BEGIN TRANSACTION and COMMIT TRANSACTION statement.

Implicit

- Transaction starts automatically once first statement of a batch is received. Must still manually COMMIT or ROLLBACK transaction.

Auto-Commit transaction Mode

Default transaction management mode of the SQL Server Database Engine

Each statement is either committed or rolled back automatically upon completion.

A syntax error will result in a batch terminating error.

- This will stop the entire batch from being executed.

A run-time error will result in a statement terminating error.

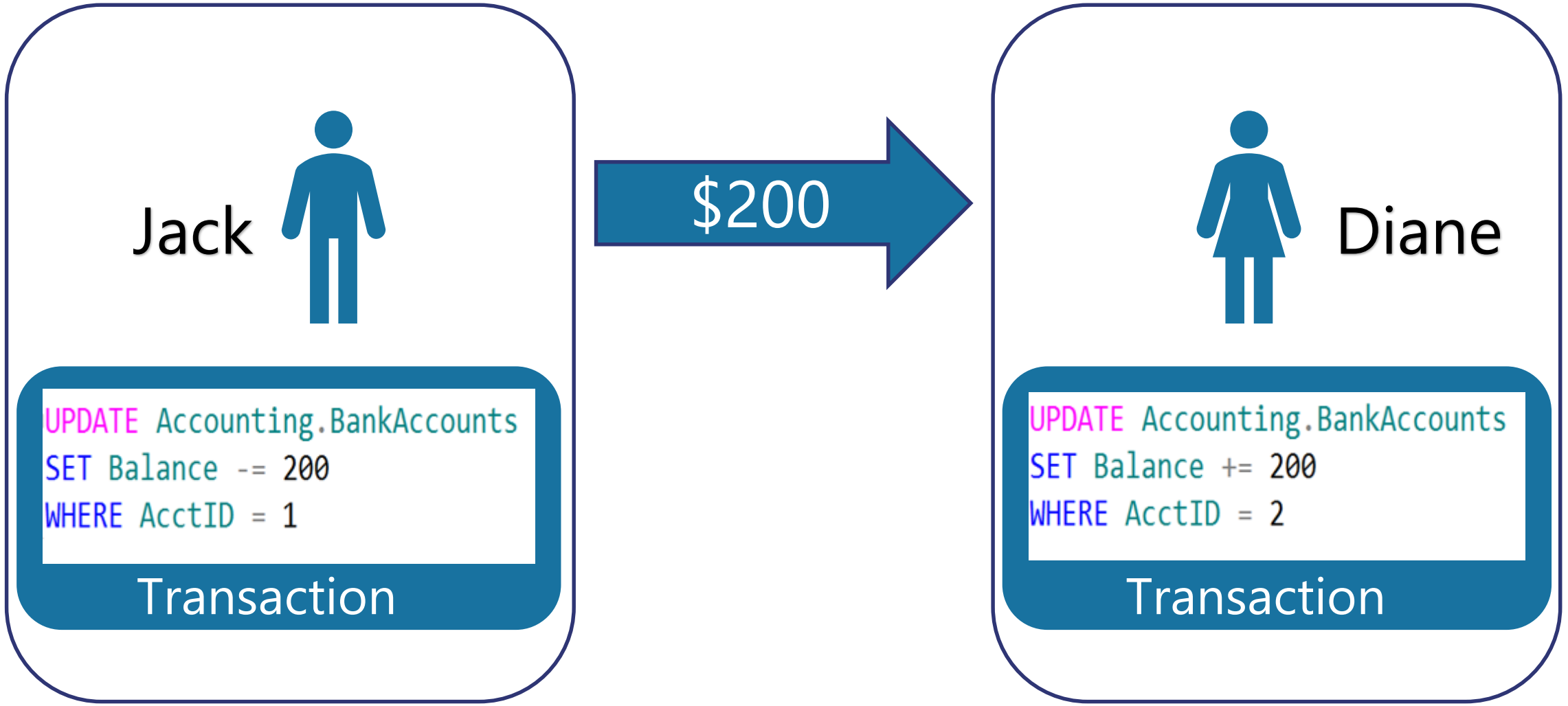
- This might allow part of the batch to commit.

Database engine operates in autocommit until started an explicit transaction

XACT_ABORT ON converts statement into batch terminating errors

Compilation errors not affected by XACT_ABORT ON

Logical Units of Work – Auto Commit Transactions



Explicit transaction mode

An explicit transaction is one in which you explicitly define both the start and end of the transaction.

Begins with the BEGIN TRANSACTION statement

Transaction can be started with a mark

Can have one or more statements

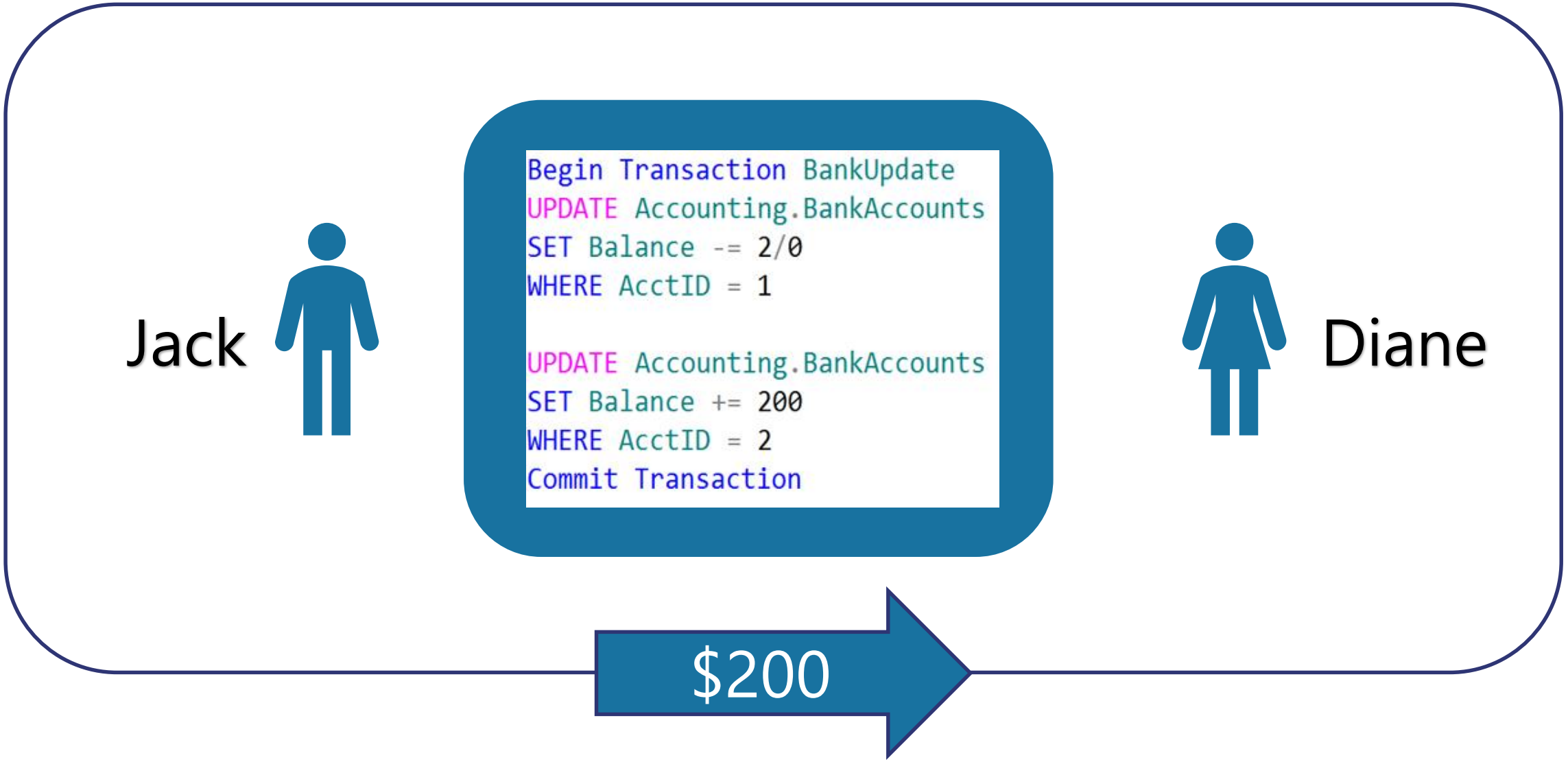
Need to explicitly commit or rollback the transaction

- using COMMIT TRANSACTION or ROLLBACK TRANSACTION

Can also use

- SAVE TRANSACTION <savepoint_name>– to rollback a transaction to named Point
- BEGIN TRANSACTION <transaction_name>WITH MARK ['*description*'] – to specify transaction marked in log

Single Logical Unit of Work – Explicit Transactions



Implicit transaction mode

- Equivalent to an unseen BEGIN TRANSACTION being executed
- Transaction starts automatically once first statement of a batch is received
- SET IMPLICIT_TRANSACTIONS ON used at statement level
- Enabled at Server level by using sp_configure 'user options' , 2
- It can be symptom of severe blocking issues on the server
- Must use commit after SELECTs or DML, otherwise transaction remains open

Considerations for using transactions

Keep transactions as short as possible

- Do not require user input
- Do not open a transaction while browsing through data
- Access the least amount of data possible
- Do not open the transaction before it is required
- Ensure that appropriate indexing is in place

Try to access resources in the same order

- Wherever possible access resources in the same order to avoid Deadlocks

Demonstration

Using IMPLICIT TRANSACTIONS and its impact on server

Knowledge Check

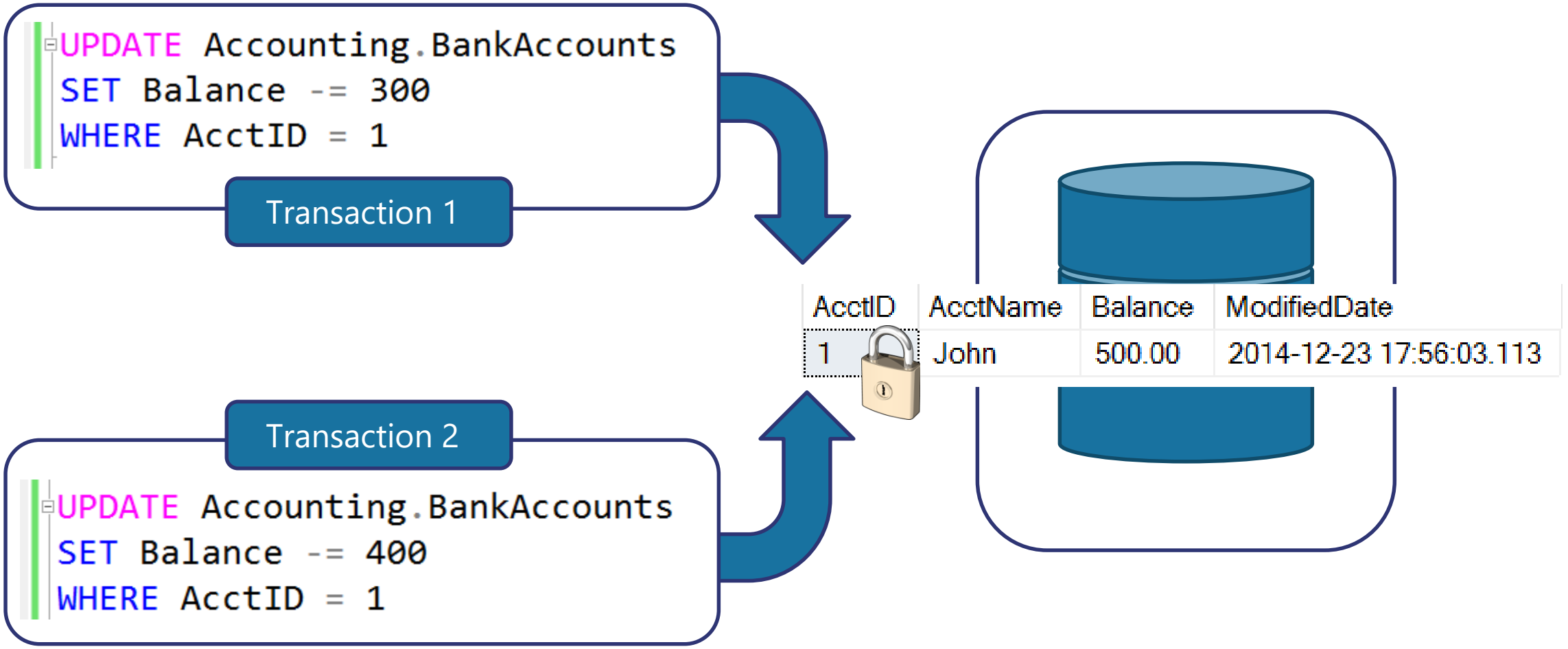
What is the default transaction mode in SQL Server?

What will the transaction count be after the following code is executed?

```
BEGIN TRANSACTION
        BEGIN TRANSACTION
ROLLBACK TRANSACTION
```

Lesson 2: SQL Server Isolation Levels

What is a Lock?



Types of concurrency problems

Following are common concurrency problems in SQL Server

Dirty Read

- The values retrieved may reflect uncommitted changes that could be rolled back.

Lost Update:

- Problem occurs when there are two transactions are unaware of each other.
- Later transaction overwrites the earlier update..

Non-Repeatable Read

- Data may change between two reads.
- If you execute a SELECT twice within a single transaction, the values returned may differ as other processes could have modify data between SELECTs.

Phantom Read

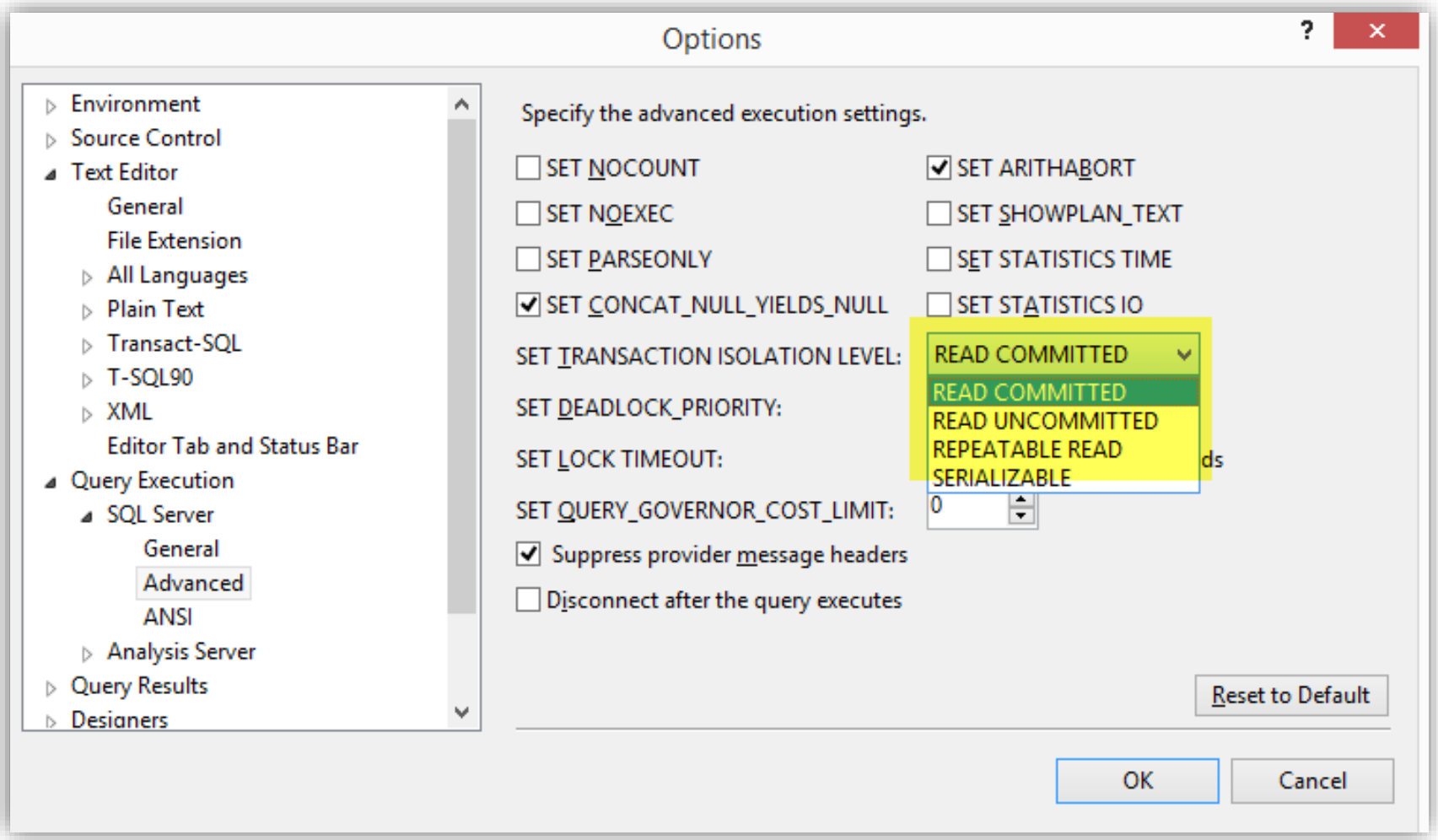
- If a range query is repeated within a single transaction, it may return additional rows not present in the first results.
- It’s still possible that another session can insert rows that belongs to the range of the initial select.

ANSI SQL Isolation Levels

Isolation is a trade-off between correctness and concurrency.

Isolation Level	Locking Duration and Range	Dirty Read	Lost Update	Non-Repeatable Read	Phantom
READ UNCOMMITTED	No locks are taken, so locks taken by other processes aren't blocking.	Yes	Yes	Yes	Yes
READ COMMITTED **(Default)**	Shared locks are taken on resources being read. They are held only for the duration of the read.	No	Yes	Yes	Yes
REPEATABLE READ	Shared locks are taken on resources being read. They are held for the duration of the transaction.	No	No	No	Yes
SERIALIZABLE	Shared range locks are taken on resources being read and adjacent resources. They are held for the duration of the transaction.	No	No	No	No

Isolation Levels



Lost Updates

```
--SQL Server Concurrency
-- Lost Update - Session 1

DECLARE @OldBalance int, @NewBalance int
BEGIN TRAN
        SELECT @OldBalance = Balance
        FROM Accounting.BankAccounts
        WHERE AcctID = 1
        SET @NewBalance = @OldBalance - 300
WAITFOR DELAY '00:00:10:000’
        UPDATE Accounting.BankAccounts
        SET Balance = @NewBalance
        WHERE AcctID = 1

        SELECT @OldBalance AS OldBalance,
        AcctID, AcctName, Balance
        FROM Accounting.BankAccounts
        WHERE AcctID = 1
COMMIT TRAN
```

OldBalance	AcctID	AcctName	Balance
500	1	John	200.00

```
--SQL Server Concurrency
-- Lost Update - Session 2

DECLARE @OldBalance int, @NewBalance int
BEGIN TRAN
    SELECT @OldBalance = Balance
    FROM Accounting.BankAccounts
    WHERE AcctID = 1
    SET @NewBalance = @OldBalance - 400

    UPDATE Accounting.BankAccounts
    SET Balance = @NewBalance
    WHERE AcctID = 1

    SELECT @OldBalance AS OldBalance,
     AcctID, AcctName, Balance
    FROM Accounting.BankAccounts
    WHERE AcctID = 1
COMMIT TRAN
```

OldBalance	AcctID	AcctName	Balance
500	1	John	100.00

Uncommitted dependency (dirty read)

```sql
--SQL Server Concurrency
-- Dirty Read - Session 1

SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED
BEGIN TRAN
    UPDATE Accounting.BankAccounts
    SET Balance -= 300
    WHERE AcctID = 1
    WAITFOR DELAY '00:00:10:000'
ROLLBACK TRAN
    SELECT AcctID, AcctName, Balance
    FROM Accounting.BankAccounts
    WHERE AcctID = 1
```

Clean Read

AcctID	AcctName	Balance	ModifiedDate
1	John	500.00	2013-02-16

```sql
--SQL Server Concurrency
--Dirty Read - Session 2

SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED
    SELECT * FROM
    Accounting.BankAccounts
    WHERE AcctID = 1
```

Dirty Read

AcctID	AcctName	Balance	ModifiedDate
1	John	200.00	2015-12-12

Inconsistent analysis (non-repeatable read)

```sql
--SQL Server Concurrency
--Repeatable Read - Session 1

SET TRANSACTION ISOLATION LEVEL
READ COMMITTED -- Before Example
--REPEATABLE READ --Switch for Example
BEGIN TRAN
    SELECT AcctID, ModifiedDate
    FROM Accounting.BankAccounts
    WAITFOR DELAY '00:00:10:000'
    SELECT AcctID, ModifiedDate
    FROM Accounting.BankAccounts
COMMIT TRAN
```

```sql
--SQL Server Concurrency
--Repeatable Read - Session 2

BEGIN TRAN
    UPDATE Accounting.BankAccounts
    SET ModifiedDate = '20130105'
COMMIT TRAN
```

READ COMMITTED

AcctID	ModifiedDate
1	2015-12-12
2	2015-12-12

AcctID	ModifiedDate
1	2013-01-05
2	2013-01-05

REPEATABLE READ

AcctID	ModifiedDate
1	2015-12-12
2	2015-12-12

AcctID	ModifiedDate
1	2015-12-12
2	2015-12-12

Phantom Reads

```
--SQL Server Concurrency
--Phantom Read - Session 1

SET TRANSACTION ISOLATION LEVEL
READ COMMITTED
BEGIN TRAN
    SELECT AcctID, AcctName,
    Balance, ModifiedDate
    FROM Accounting.BankAccounts
WAITFOR DELAY '00:00:10:000'
    SELECT AcctID, AcctName,
    Balance, ModifiedDate
    FROM Accounting.BankAccounts
COMMIT TRAN
```

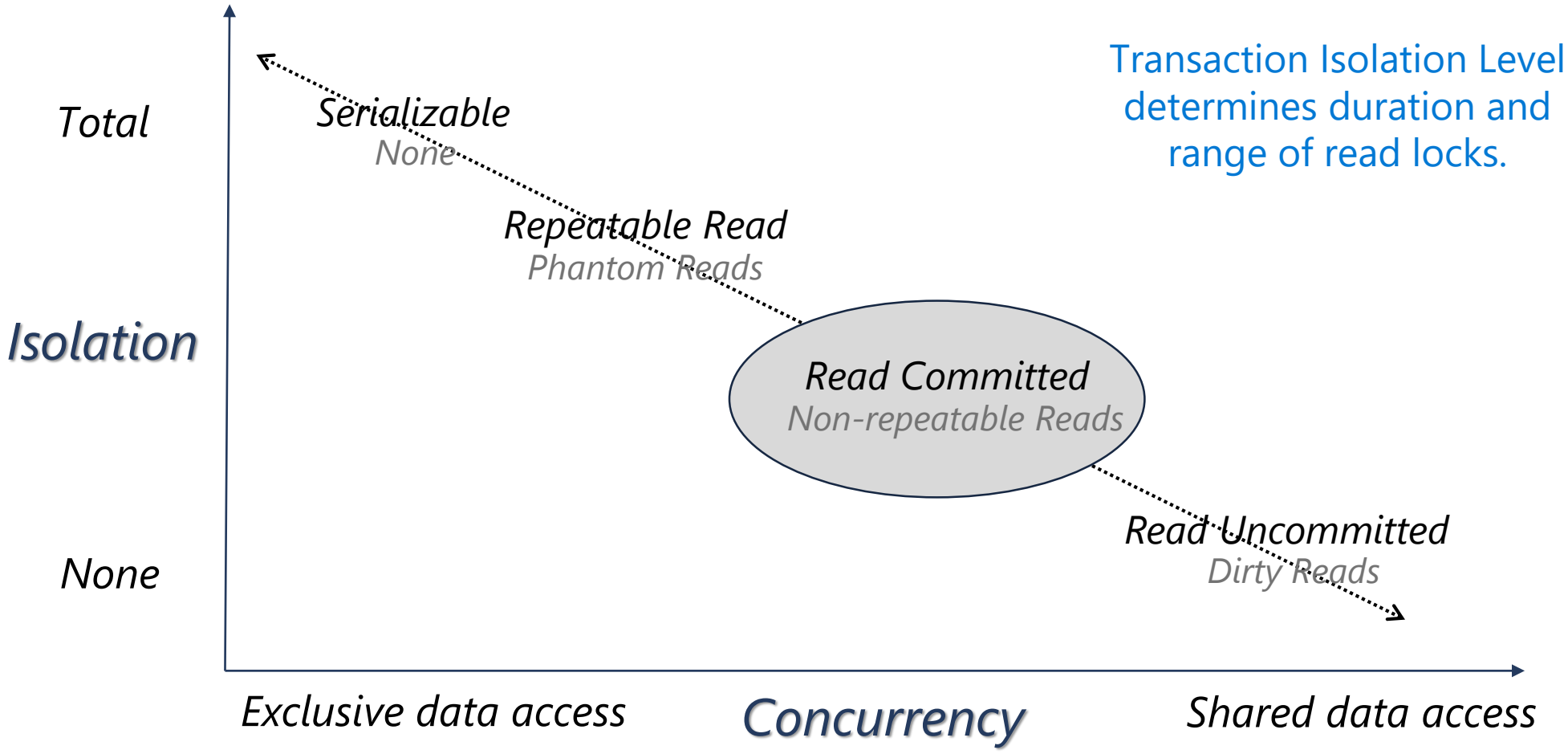
```
--SQL Server Concurrency
--Phantom Read - Session 2
USE TestDB
BEGIN TRAN
DELETE FROM Accounting.BankAccounts
WHERE AcctID IN(12, 15, 21)
COMMIT TRAN
```

AcctID	AcctName	Balance	ModifiedDate
1	Jack	500.00	2013-01-05
2	Diane	750.00	2013-01-05
12	Danny	630.00	2013-01-05
14	Mayleigh	204.00	2013-01-05
15	Molly	790.00	2013-01-05
18	Maddison	745.00	2013-01-05
21	Alicen	555.00	2013-01-05
22	Bella	445.00	2013-01-05

Missing records

AcctID	AcctName	Balance	ModifiedDate
1	Jack	500.00	2013-01-05
2	Diane	750.00	2013-01-05
14	Mayleigh	204.00	2013-01-05
18	Maddison	745.00	2013-01-05
22	Bella	445.00	2013-01-05
23	Logan	1050.00	2013-01-05

Balancing Isolation with concurrency



Delayed Durability

- Accomplished using asynchronous log writes to disk.
- Returns control to the client application before the transaction log buffer is flushed.
- Reduces both latency and contention within the system.
- Slight risk of data loss if the server crashed before the log buffer is flushed.

Transaction Level Setting	Database Level Setting
BEGIN TRANSACTION --Some DML statement COMMIT { TRAN | TRANSACTION } [transaction_name | @tran_name_variable]] WITH (DELAYED_DURABILITY = { OFF|ON })	ALTER DATABASE <Db_Name> SET DELAYED_DURABILITY = { DISABLED (default) | ALLOWED | FORCED }

Concurrency

Avoids inconsistency result or abnormal behavior

Support for two concurrency models

- Pessimistic concurrency
  - This is the default concurrency model in SQL Server
  - Uses locks to avoid concurrency problems
- Optimistic concurrency
  - Uses row versions to support concurrency

Snapshot Isolation

- First Introduced in SQL Server 2005.
- Uses Optimistic concurrency.
- Managed using Row Versioning.
- If ADR is not enabled, row versions are kept in tempdb.

Read Committed Snapshot Isolation (RCSI)	Snapshot Isolation
Can be used with Distributed Transaction	Can't be used with Distributed Transaction
No code changes required to manage conflicts	Code changes required to manage conflicts
Statement Level Consistency	Transaction-level consistency
Writers still block other writers	Writers don't block writers, only the first to commit wins, others must rollback/retry
No update conflict	Prone to update conflict

Enabling row versioning-based isolation

- Read committed snapshot Isolation (RCSI)
  - READ_COMMITTED_SNAPSHOT database enabled plus READ Committed isolation level

```
ALTER DATABASE MyDatabase SET READ_COMMITTED_SNAPSHOT ON
```

- Snapshot Isolation
  - ALLOW_SNAPSHOT_ISOLATION database enabled plus SNAPSHOT isolation level

```
ALTER DATABASE MyDatabase SET ALLOW_SNAPSHOT_ISOLATION ON
```

- Prevents readers from blocking writers
- Uses TempDB and avoids placing shared Locks

Demonstration

Key range locking

Using sys.dm_tran_locks to understand the impact of Locked resources and duration according to Isolation Level

SQL Server Isolation Levels

Illustrate how snapshot isolation level effect blocking and locking

Knowledge Check

What is the default isolation level?

What isolation levels can be set at the database level?

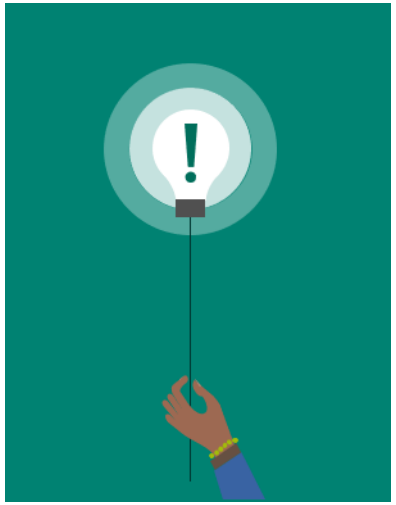
What could be the impact of using the sys.dm_tran_version_store DMV?

Lesson 3: SQL Server Locking

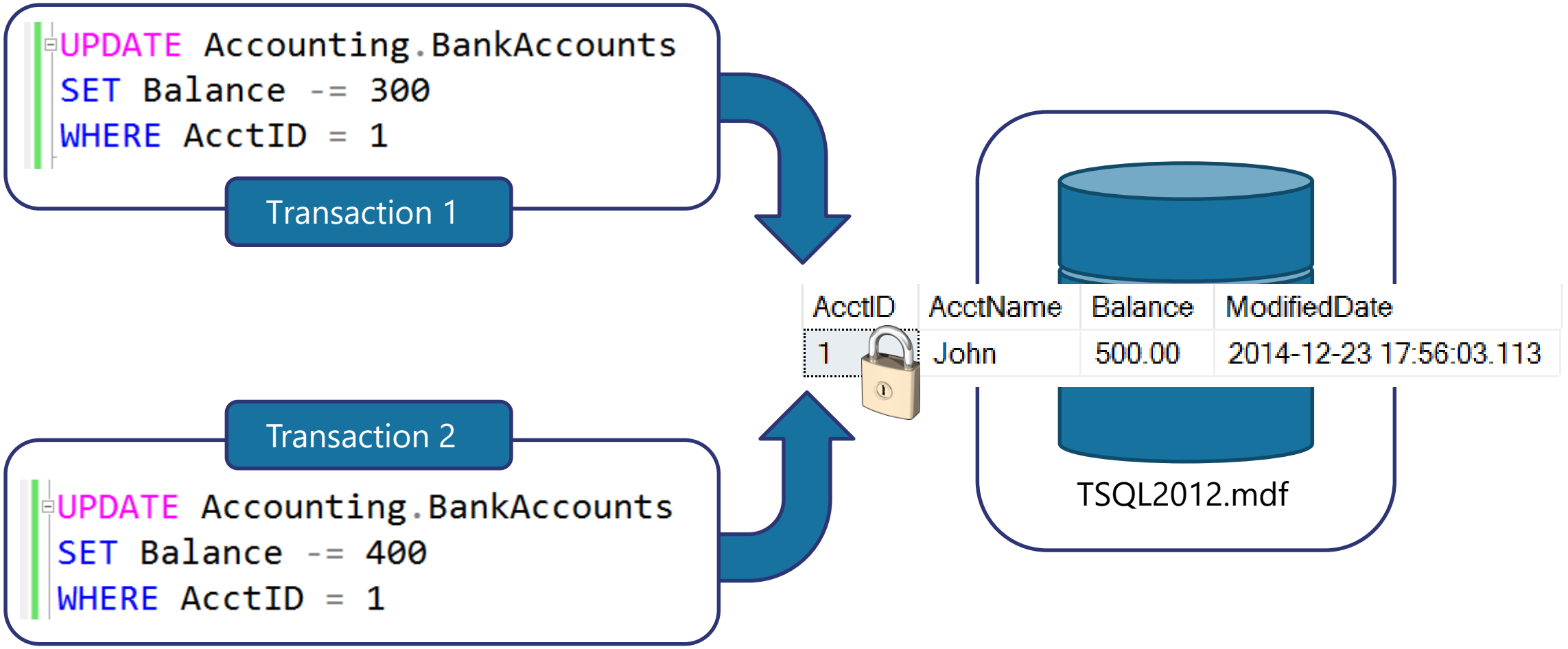
Objectives

After completing this learning, you will be able to:

- Understand locking concepts.
- Understand lock Modes and lock compatibility.
- How SQL server choose granularity.
- Learn how SQL server escalate locks.
- What are lock Hints ?

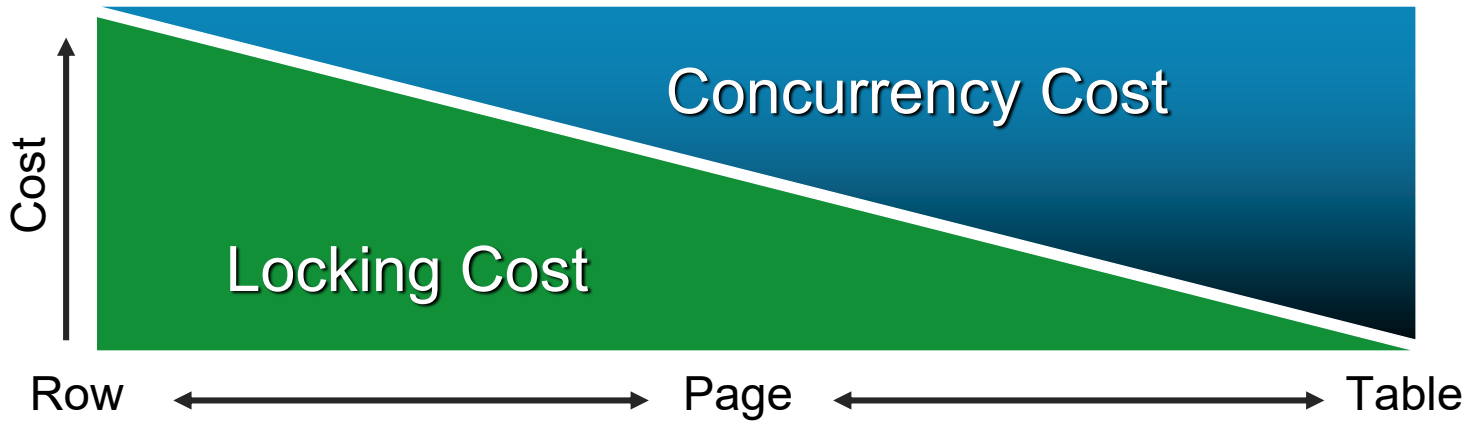


What is a Lock?



Multi-Granular Locking

- Many items can be locked in SQL Server
  - Databases
  - Schema
  - Objects
- Some objects can be locked at different levels of granularity
- SQL Server will automatically choose the granularity of the lock based on the estimated cost
- Multiple levels of granularity are grouped into a lock hierarchy



Lock Granularity and Hierarchies

Resource	Description
RID	A row identifier used to lock a single row within a heap.
KEY	A row lock within an index used to protect key ranges in serializable transactions.
PAGE	An 8-kilobyte (KB) page in a database, such as data or index pages.
EXTENT	A contiguous group of eight pages, such as data or index pages.
HoBT	A heap or B-tree. A lock protecting heap data pages in a table that does not have a clustered index or the pages of a B-tree index.
TABLE	The entire table, including all data and indexes.
FILE	A database file.
ALLOCATION_UNIT	An allocation unit.
DATABASE	The entire database.

Lock Duration

Mode	Read Committed	Repeatable Read	Serializable	Snapshot
Shared	Held until data read and processed	Held until end of transaction	Held until end of transaction	N/A
Update	Held until data read and processed unless promoted to Exclusive	Held until data read and processed unless promoted to Exclusive	Held until end of transaction unless promoted to Exclusive	Held until data read and processed unless promoted to Exclusive
Exclusive	Held until end of transaction	Held until end of transaction	Held until end of transaction	Held until end of transaction

Lock Mode - Standard

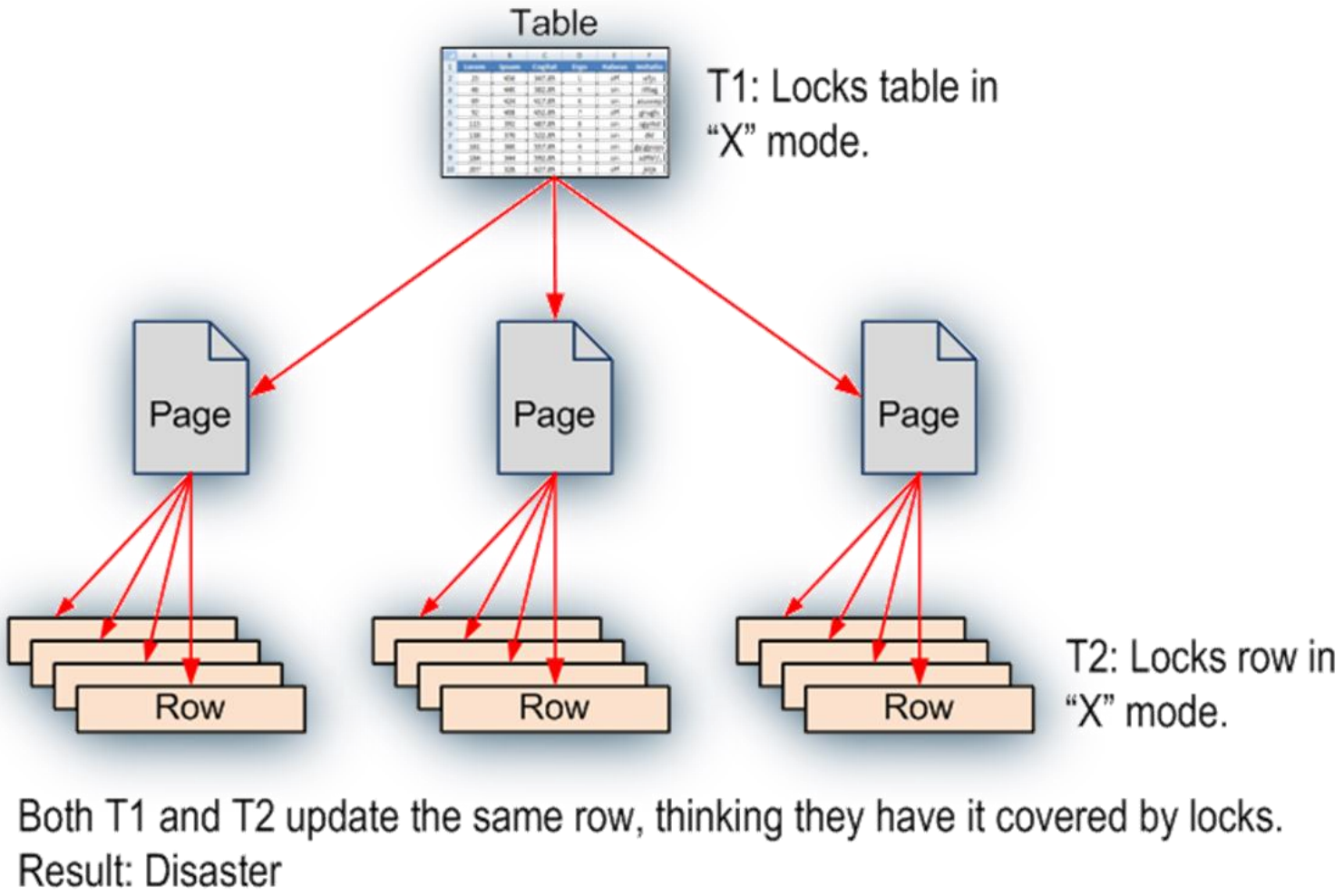
Lock Mode	Description
Schema-Stability (Sch-S)	Used when compiling queries
Schema Modification (Sch-M)	Used when a table data definition language operation (for example, dropping a table) is being performed
Shared (S)	Used for read operations that do not change or update data, such as a SELECT statement
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 later
Exclusive (X)	Used for data-modification operations, such as INSERT, UPDATE, or DELETE. Ensures that multiple updates cannot be made to the same resource at the same time

Lock Mode - Special

Lock Mode	Description
Intent Shared (IS)	Have or will request shared lock(s) at a finer level
Intent Update (IU)	Have or will request update lock(s) at a finer level
Intent Exclusive (IX)	Have or will request exclusive lock(s) at a finer level
Shared Intent Update (SIU)	Have shared lock with intention to acquire update lock at a finer level
Shared Intent Exclusive (SIX)	Have shared lock with intention to acquire exclusive lock at a finer level
Update Intent Exclusive (UIX)	Have update lock with intention to acquire exclusive lock at a finer level
Bulk Update (BU)	Used when bulk copying data into a table and either TABLOCK hint is specified or the table lock on bulk load table option is set

Establish Lock Hierarchy with Intent Locks

To acquire a fine granular lock, you must acquire intent locks on all the higher levels in the hierarchy



Lock hierarchy with intent locks

SQL Server uses intent locks to protect parent-level object in the hierarchy by placing an intent shared (IS) or Intent exclusive (IX) lock.

Intent locks are acquired before a lock placed at the lower level.

Intent locks serve two purposes:

- Prevent other transactions from modifying parent-level object
- Improve the efficiency of the SQL Server Database Engine

Lock Compatibility

	Existing Granted Mode					
Requested Mode	**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

Dynamic locking

Row locking is not always the right choice

- Scanning 100 million rows means 100 million calls to the lock manager

Page, Partition or Table locking can be more efficient

- One Table lock is cheaper and easier to manage than thousands of Row locks

SQL Server chooses lock granularity (Row, Page, Table) at run time based on input from the Query Optimizer

- Least-expensive method is chosen
- Available resources at the time of execution may have an impact
- Incorrect estimates could lead to making the wrong choice

Lock escalation

Lock manager attempts to replace many row or page locks with a single table-level lock.

- One Table lock is faster and easier to manage than thousands of Row locks.
- One Table lock requires less memory than many Row Locks.

It never converts row locks to page locks.

Lock de-escalation never occurs.

Lock Escalation behavior can be controlled.

Server Level	Table Level
Trace Flag 1211 Disables lock escalation due to memory pressure	AUTO | TABLE (Default) | DISABLE

Syntax:

```
ALTER TABLE Table_name  SET ( LOCK_ESCALATION = { AUTO | TABLE | DISABLE } )
```

Locking Hints

- Override the default behavior of the query optimizer
- Table hints are specified in the **FROM clause** of the DML statement
- Affect only the table or view referenced in that clause
- Locking method can be used at various levels as shown below

Granularity Level hints	ROWLOCK, PAGLOCK and TABLOCK
Isolation LEVEL hints	HOLDLOCK/SERIALIZABLE, NOLOCK/ READUNCOMMITTED, READCOMMITTED,REPEATABLEREAD, READCOMMITTED
UPDLOCK	Use update lock rather than shared lock when reading
XLOCK	Use exclusive lock instead
READPAST	Skips currently locked rows

Demonstration

SQL Server Multi-granular locking

- Using DMVs to obtain lock information
- Using Extended Events to capture Lock Escalation

Knowledge Check

What is a Lock?

Why is an intent lock acquired?

What is lock escalation and can it be controlled?

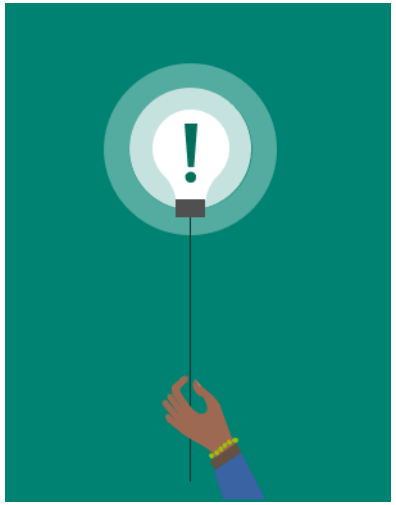
What are locking hints?

Lesson 4: Troubleshooting Concurrency Performance

Objectives

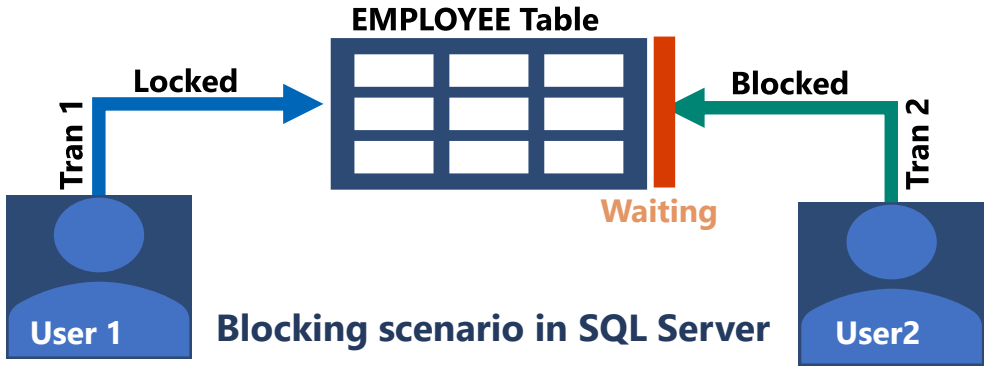
After completing this lesson, you will be able to:

- Describe blocking concepts.
- Troubleshoot blocking problems.
- Explain deadlock concepts.
- Monitor, analyze and resolve deadlock occurrences.



Blocking

Blocking is an unavoidable characteristic of any RDBMS with lock-based concurrency.



Capturing blocking Information

- A Custom SQL scripts using DMVs that monitor locking and blocking
- Use SSMS standard reports i.e. Activity – All blocking transactions
- Extended events - blocked process report

Common Blocking Scenarios and Resolution

- Long running Query
- Orphaned connection
- SPID whose corresponding client application did not fetch all result rows to completion
- Sleeping server SPID that has lost track of the Transaction Nesting Level
- Inappropriate transaction or transaction-isolation level
- SPID that is in a rollback state
- Distributed client/server deadlock

Detecting Compile Blocking

Dynamic views

- sys.dm_exec_requests
- sys.dm_tran_locks

Performance Monitor

- Possible High CPU utilization

Resolution:

- Check to see for two-part name
- Troubleshoot stored procedure recompile
- Avoid procedure names with sp_

Minimizing Blocking

- Keep transactions short and in one batch.
- Avoid user interaction in a transaction.
- Use proper indexing – The Database Tuning Advisor index analysis.
- Beware of the implicit transactions.
- Reduce the isolation level to lowest possible.
- Locking hint, Index hint, Join hint.
- Roll back when canceling; Roll back on any error or timeout.
- Apply a stress test at maximum projected user load before deployment.

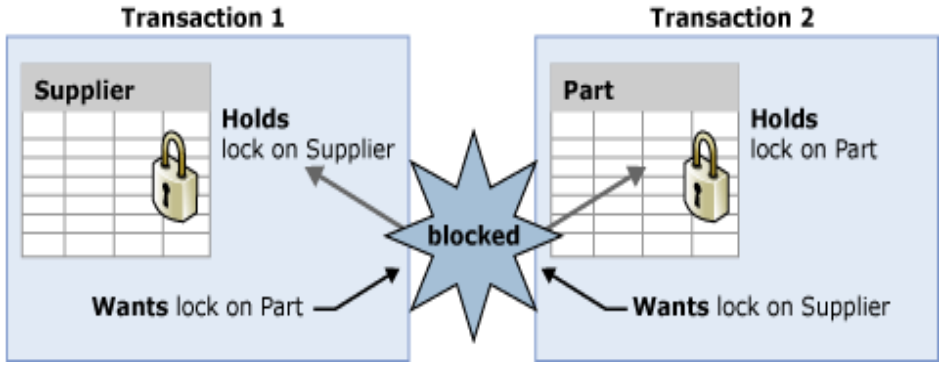
Demonstration

View lock blocking

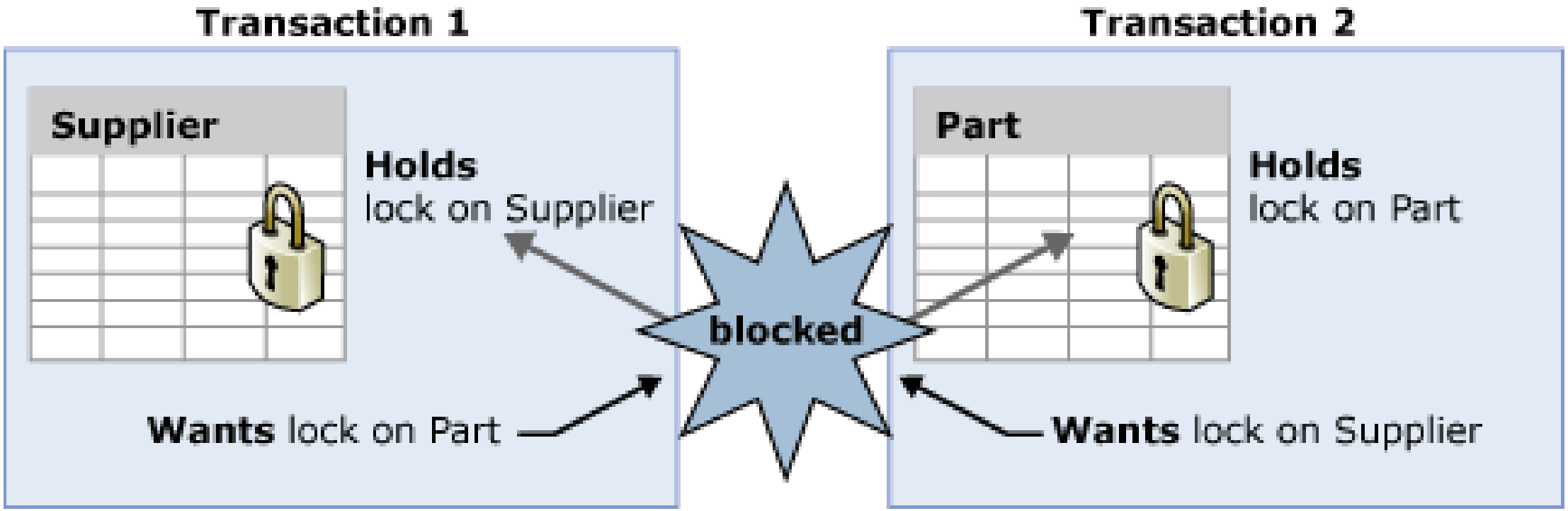
What is a deadlock?

Two or more processes are waiting for one another to obtain locked resources

- Transaction 1 **holds** a lock on the Supplier table and **requires a lock** on the Part table.
- Transaction 2 **holds** a lock on the Part table and **wants a lock** on the Supplier table
- Both tasks cannot continue until a resource is available and the resources cannot be released until a task continues, and therefore a deadlock state exists



What Is a Deadlock?



Detection and Identification

Lock monitor thread periodically perform deadlock detection in SQL Server.

Trace flags 1222,1204.

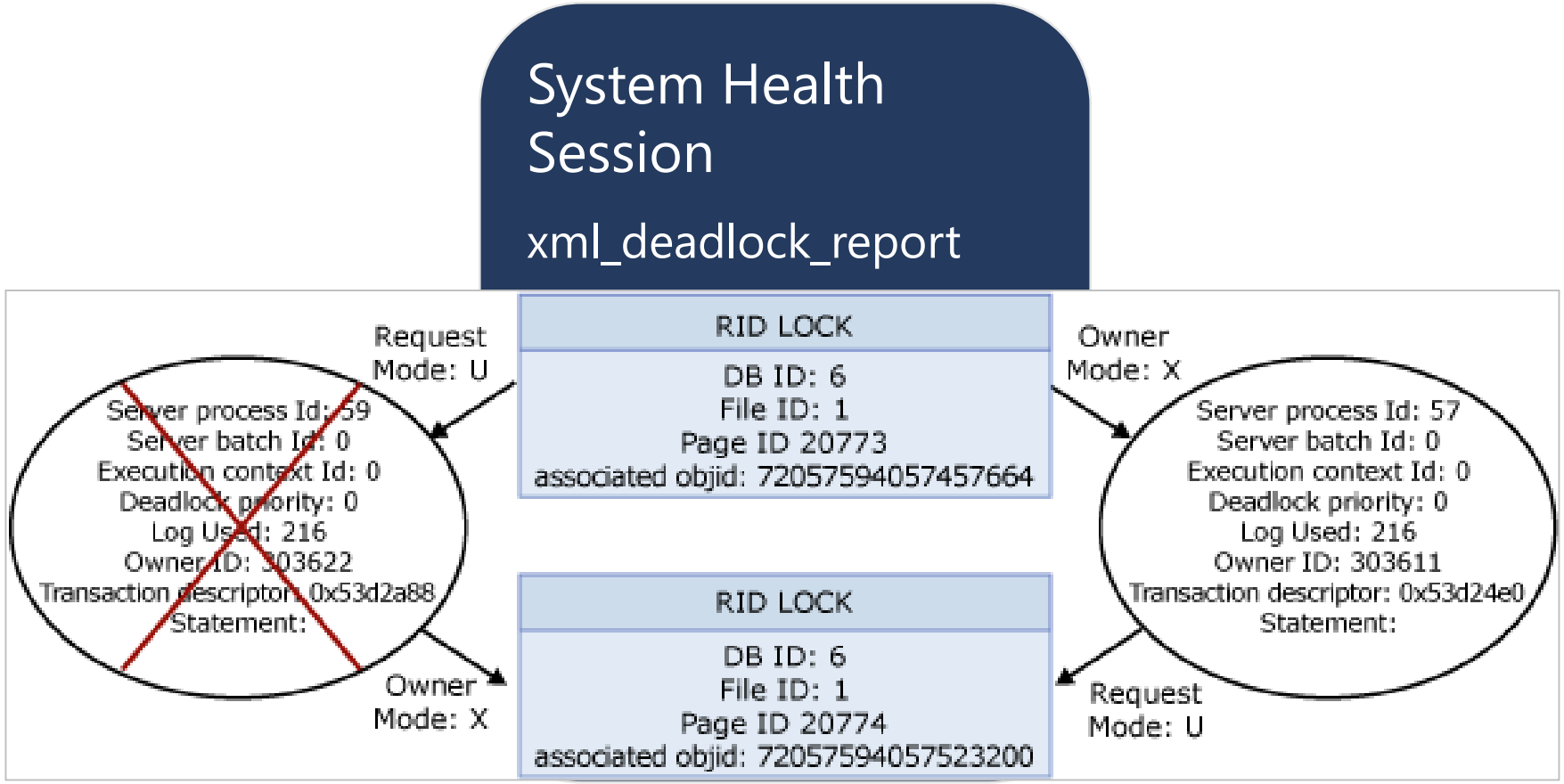
Deadlocks are captured by the system health event session.

When Deadlock occurs, SQL server returns 1205 error code to application.

- Applications should use retry logic.
- Check for 1205 error code to resubmit the transaction.

Deadlock Analysis

Using System Health Xevent



Deadlock Analysis

Using Trace Flags

Use Trace Flag 1222 or 1204 to write deadlock information in SQL Server Error log.

Trace Flag 1222 Example

```
deadlock-list
 deadlock victim=process689978
  process-list
   process id=process6891f8 taskpriority=0 logused=868
   waitresource=RID: 6:1:20789:0 waittime=1359 ownerId=310444
   transactionname=user_transaction
   lasttranstarted=2005-09-05T11:22:42.733 XDES=0x3a3dad0
   lockMode=U schedulerid=1 kpid=1952 status=suspended spid=54
   sbid=0 ecid=0 priority=0 transcount=2
   lastbatchstarted=2005-09-05T11:22:42.733
   lastbatchcompleted=2005-09-05T11:22:42.733
   clientapp=Microsoft SQL Server Management Studio - Query
   hostname=TEST_SERVER hostpid=2216 loginname=DOMAIN\user
   isolationlevel=read committed (2) xactid=310444 currentdb=6
   lockTimeout=4294967295 clientoption1=671090784 clientoption2=390200
    executionStack
     frame procname=AdventureWorks2008R2.dbo.usp_p1 line=6 stmtstart=202
     sqlhandle=0x0300060013e6446b027cbb00c6960000010000000000000
     UPDATE T2 SET COL1 = 3 WHERE COL1 = 1;
     frame procname=adhoc line=3 stmtstart=44
     sqlhandle=0x01000600856aa70f503b8104000000000000000000000000
     EXEC usp_p1
    inputbuf
      BEGIN TRANSACTION
        EXEC usp_p1
  process id=process689978 taskpriority=0 logused=380
   waitresource=KEY: 6:72057594057457664 (350007a4d329)
   waittime=5015 ownerId=310462 transactionname=user_transaction
   lasttranstarted=2005-09-05T11:22:44.077 XDES=0x4d9e258 lockMode=U
   schedulerid=1 kpid=3024 status=suspended spid=55 sbid=0 ecid=0
   priority=0 transcount=2 lastbatchstarted=2005-09-05T11:22:44.077
   lastbatchcompleted=2005-09-05T11:22:44.077
   clientapp=Microsoft SQL Server Management Studio - Query
   hostname=TEST_SERVER hostpid=2216 loginname=DOMAIN\user
   isolationlevel=read committed (2) xactid=310462 currentdb=6
   lockTimeout=4294967295 clientoption1=671090784 clientoption2=390200
    executionStack
     frame procname=AdventureWorks2008R2.dbo.usp_p2 line=6 stmtstart=200
```

Trace Flag 1204 Example

```
Deadlock encountered .... Printing deadlock information
Wait-for graph

Node:1

RID: 6:1:20789:0              CleanCnt:3 Mode:X Flags: 0x2
 Grant List 0:
   Owner:0x0315D6A0 Mode: X
     Flg:0x0 Ref:0 Life:02000000 SPID:55 ECID:0 XactLockInfo: 0x04D9E27C
   SPID: 55 ECID: 0 Statement Type: UPDATE Line #: 6
   Input Buf: Language Event:
BEGIN TRANSACTION
   EXEC usp_p2
 Requested By:
   ResType:LockOwner Stype:'OR'Xdes:0x03A3DAD0
     Mode: U SPID:54 BatchID:0 ECID:0 TaskProxy:(0x04976374) Value:0x315d200 Cost:(0/868)

Node:2

KEY: 6:72057594057457664 (350007a4d329) CleanCnt:2 Mode:X Flags: 0x0
 Grant List 0:
   Owner:0x0315D140 Mode: X
     Flg:0x0 Ref:0 Life:02000000 SPID:54 ECID:0 XactLockInfo: 0x03A3DAF4
   SPID: 54 ECID: 0 Statement Type: UPDATE Line #: 6
   Input Buf: Language Event:
     BEGIN TRANSACTION
       EXEC usp_p1
 Requested By:
   ResType:LockOwner Stype:'OR'Xdes:0x04D9E258
     Mode: U SPID:55 BatchID:0 ECID:0 TaskProxy:(0x0475E374) Value:0x315d4a0 Cost:(0/380)

Victim Resource Owner:
 ResType:LockOwner Stype:'OR'Xdes:0x04D9E258
     Mode: U SPID:55 BatchID:0 ECID:0 TaskProxy:(0x0475E374) Value:0x315d4a0 Cost:(0/380)
```

Resolution and Avoidance

SQL Server automatically selects the transaction that is the cheapest to roll back.

Use SET DEADLOCK_PRIORITY to change the likelihood of a batch being chosen as a victim.

Access the resources in the same order.

Make the transactions simple and shorten the length of the transactions.

Use error handlers to capture deadlocks.

Demonstration

Deadlocks

Troubleshooting Concurrency Performance

- Identify blocking using DMVs.
- Identify blocking using Extended Events.

Knowledge Check

How is a deadlock detected?

How should a deadlock be handled in an application?

What trace flags are used in a deadlock analysis?

What is the Extended Event used in a deadlock analysis?