Microsoft

SQL Server Concurrency and Transactions

Module 3

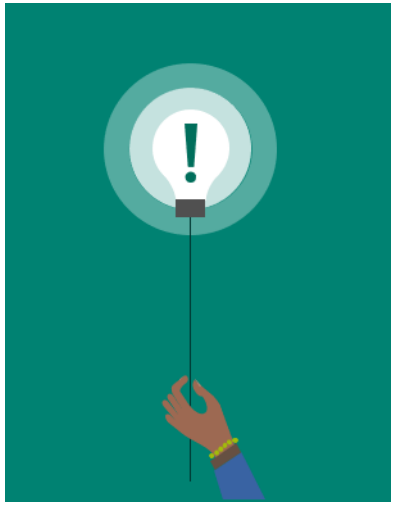
Learning Units covered in this Module

- Lesson 1: SQL Server Transactions
- Lesson 2: SQL Server Locking

Objectives

After completing this learning, you will be able to:

- Understand what is a transaction and its types.
- Recommendations for using transactions.
- Describe the concurrency and concurrency types.
- Blocking and Deadlocking



Lesson 1: SQL Server 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.

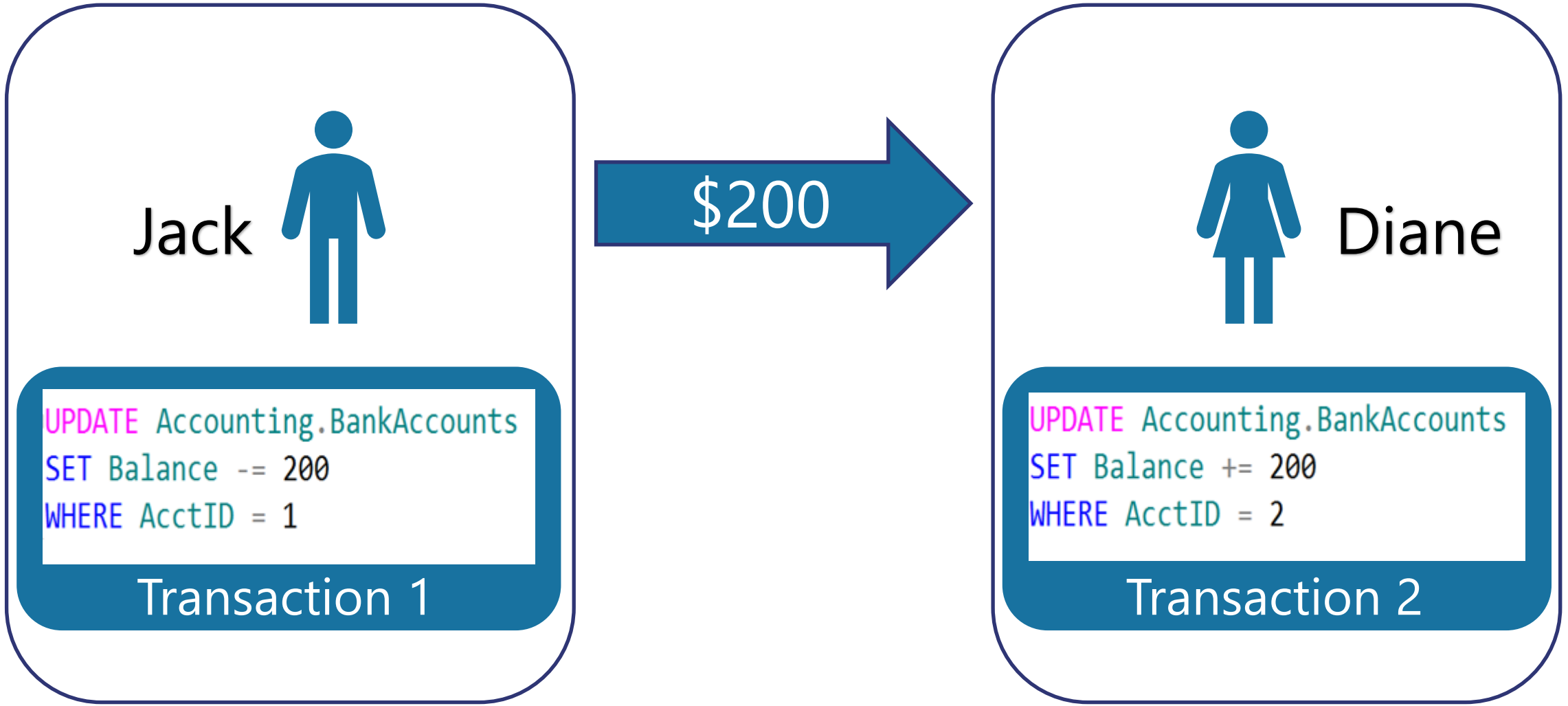
- This might allow part of the batch to commit.

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

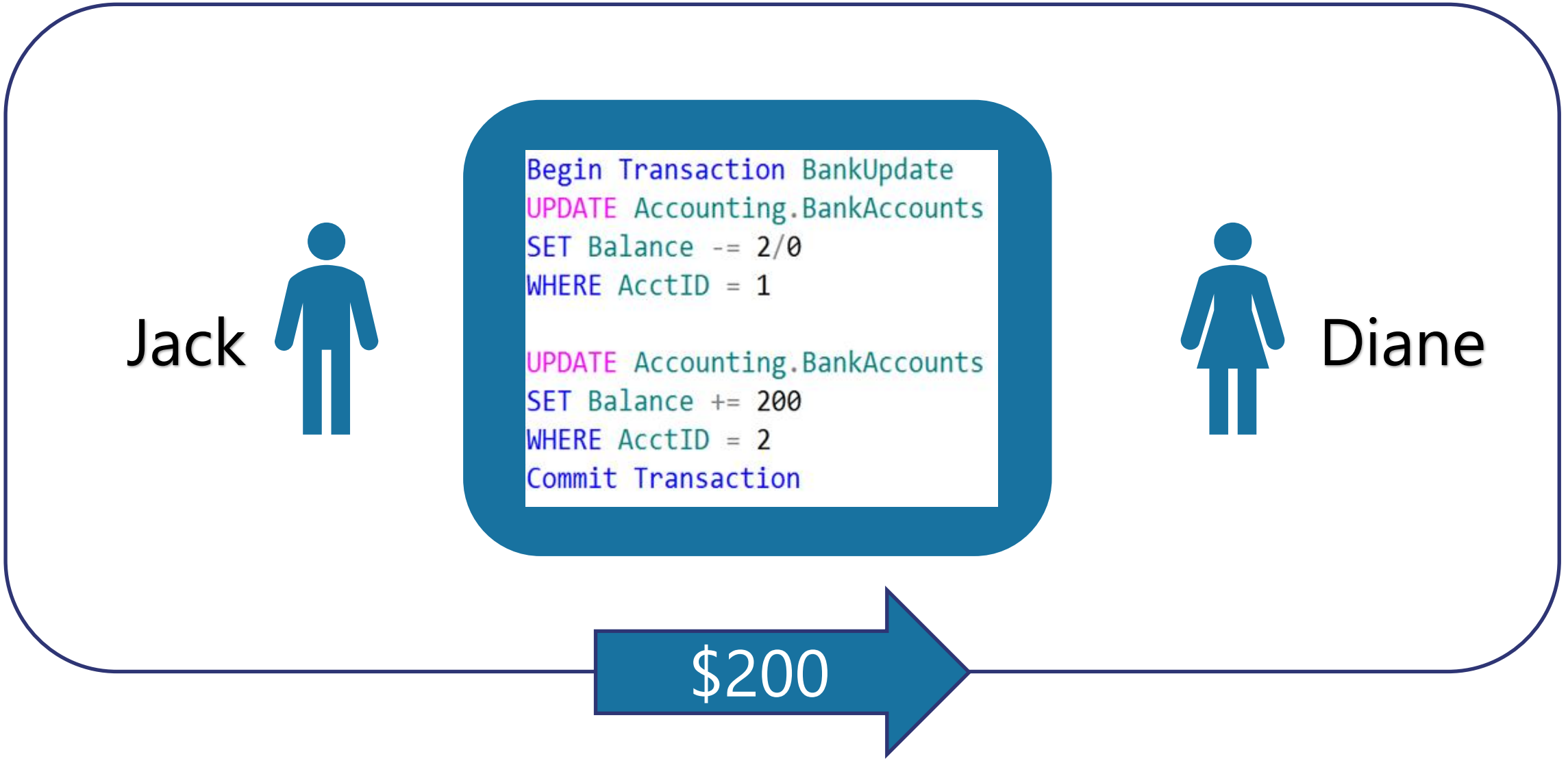
Need to explicitly commit or rollback the transaction

- 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

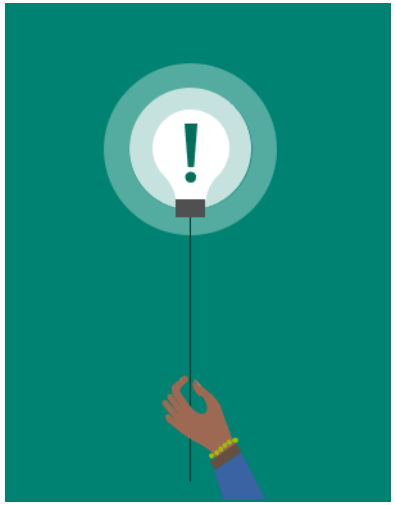
- Wherever possible access resources in the same order to avoid Deadlocks

Lesson 2: SQL Server Locking

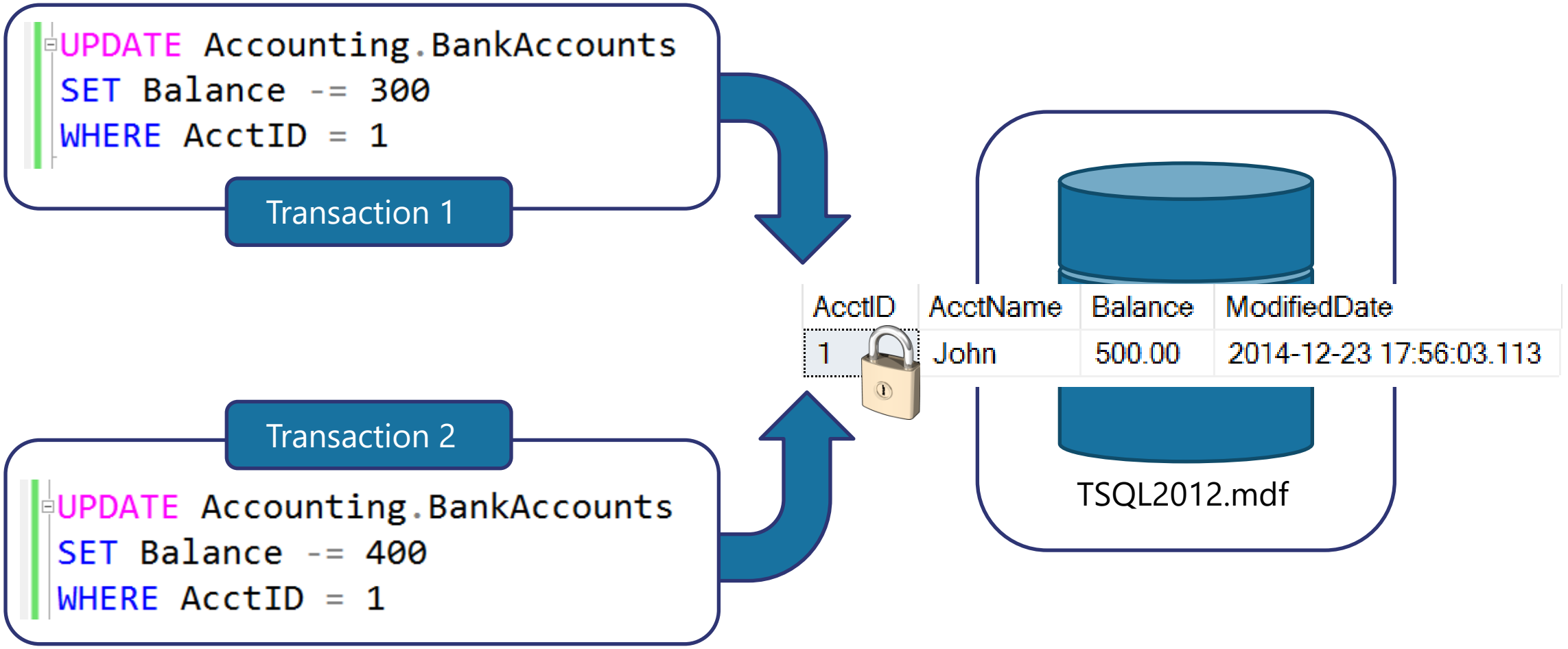
Objectives

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 ?

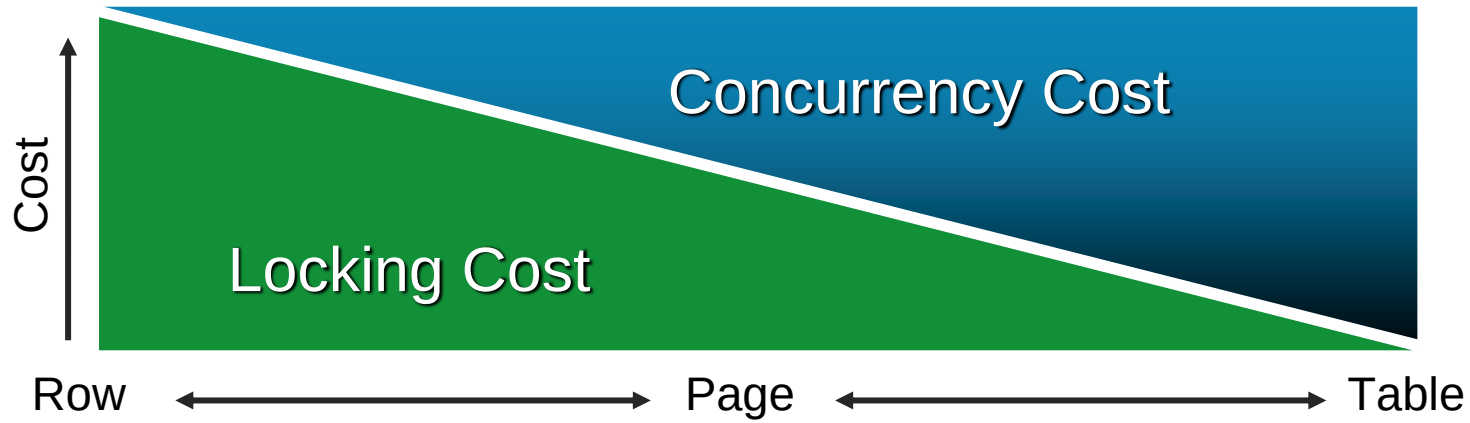


What is a Lock?



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

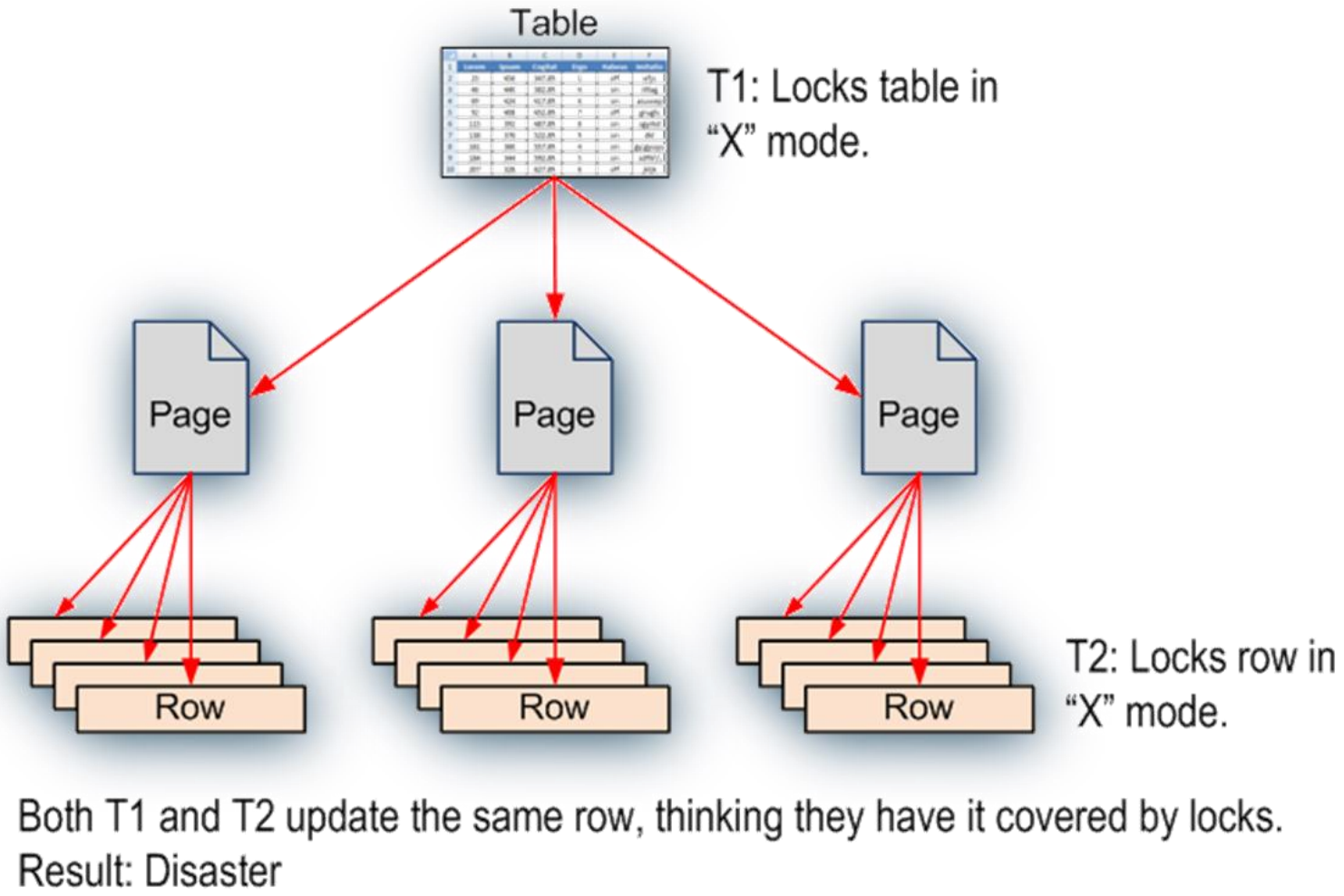
Lock Mode	Description
Schema-Stability (Sch-S)	Used when compiling queries
Schema Modification (Sch-M)	Used when a table data definition language operation (for example, dropping a table) is being performed
Shared (S)	Used for read operations that do not change or update data, such as a SELECT statement
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?

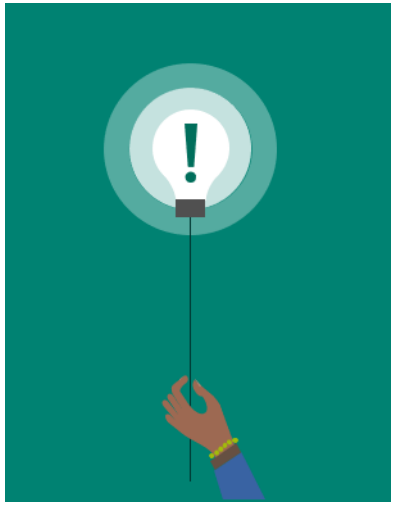
What are locking hints?

Lesson 3: Blocking and Deadlocking

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.

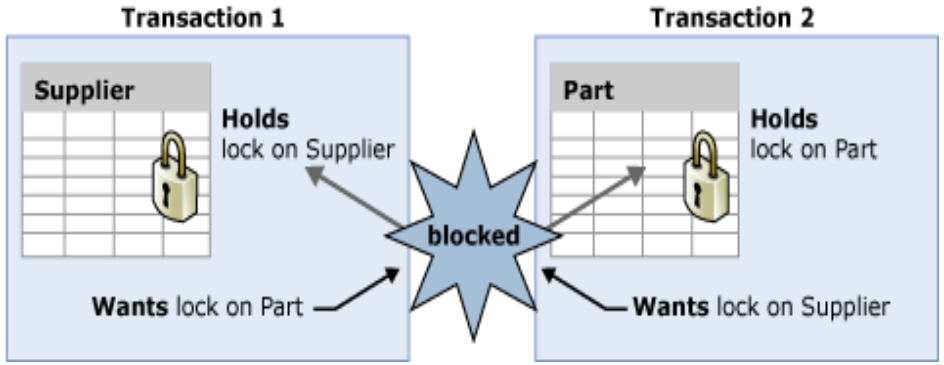
Capturing blocking Information

- 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

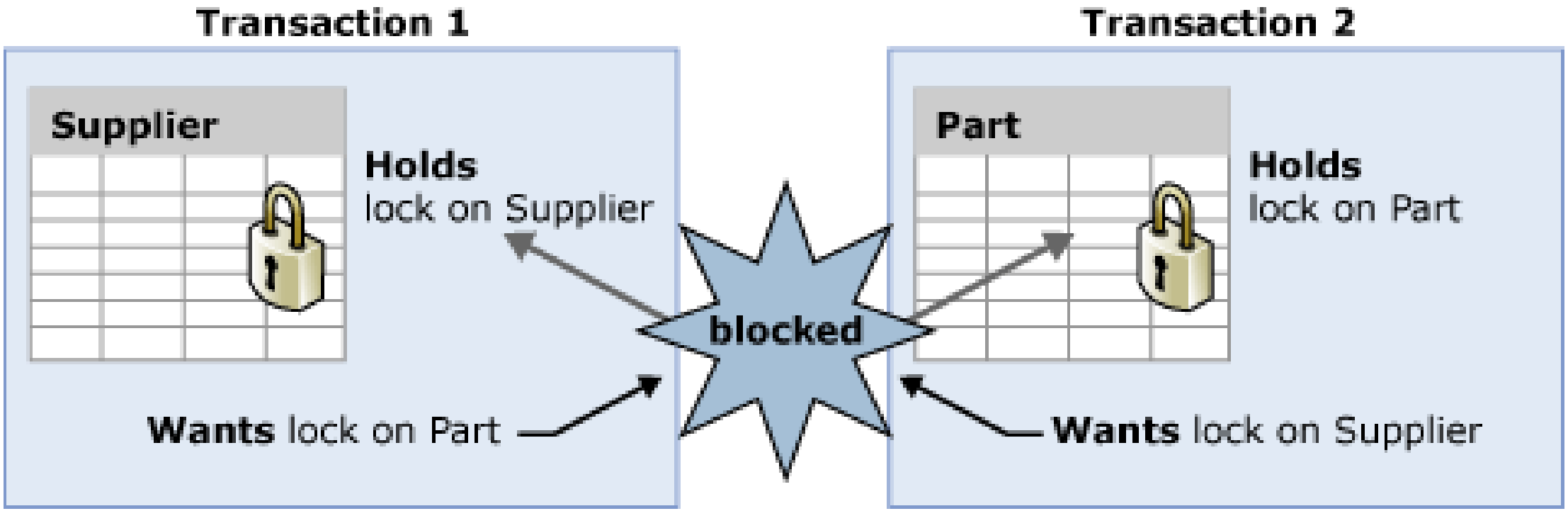
What is a deadlock?

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

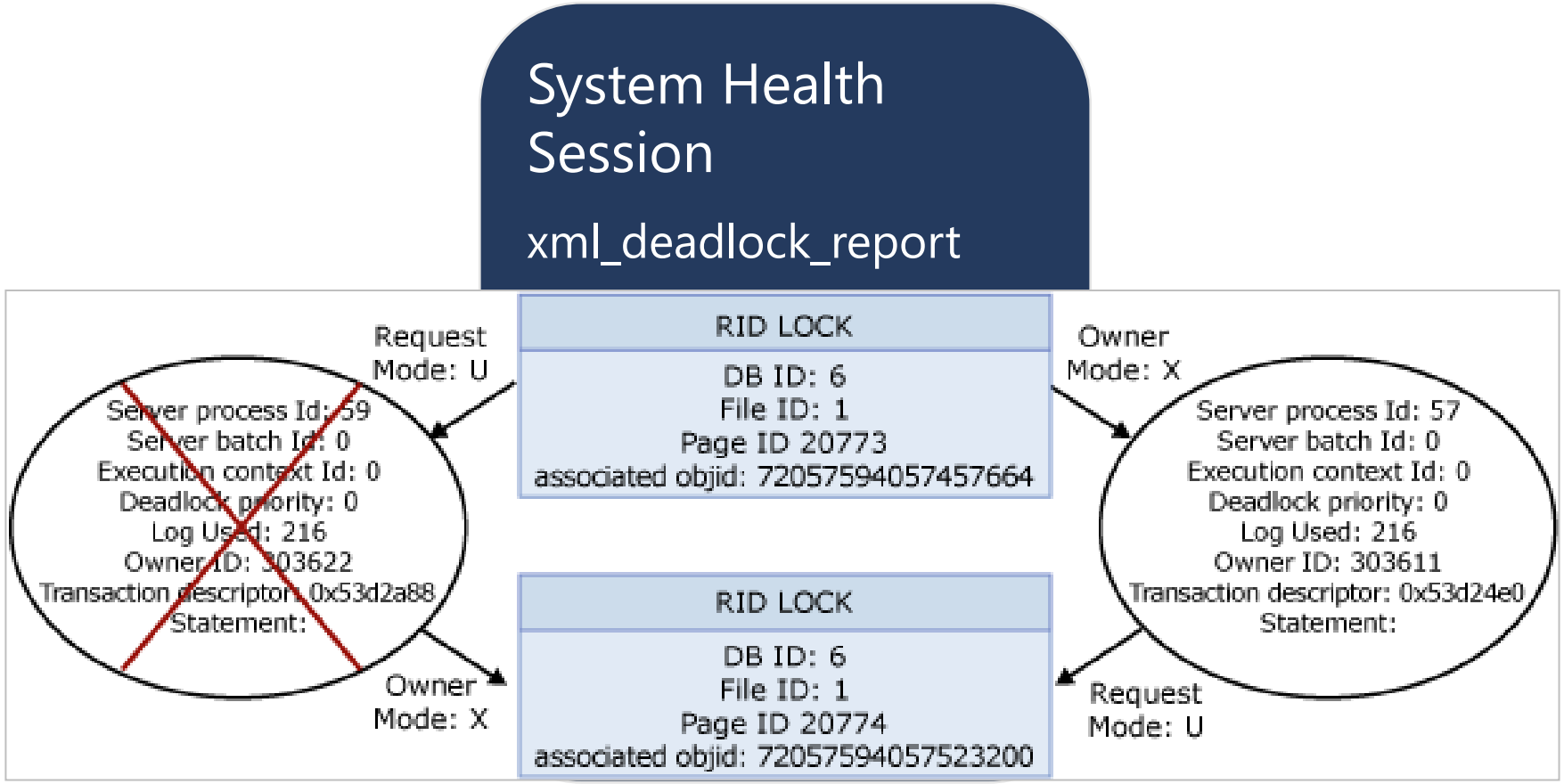


What Is a Deadlock?



Deadlock Analysis

Using System Health Xevent



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?