Content of the Book: Details both concurrency models and explain what factors we must consider when comparing the relative costs on our systems

Session: is a single connection to the SQL Server. Has a SessionID  
Locking: claims ownership over an resource.  
Blocking: occurs when at least two sessions desire concurrent access to the same resource.  
Deadlock: occurs when two sessions mutually block each other.  
Pressure: is a term used to indicate a state where competition for access to a certain resource is causing performance issues.

# Chapter 1: Concurrency and Transactions

Two models: pessimistic and optimistic.

By default, SQL Server adopts a pessimistic approach to concurrency.

Conflicts, when multiple sessions try to modify the same data at the same time. In such situations, several resource contention and data integrity issues can arise, such as:

* Preventable read phenomena
  + dirty read
  + non-repeatable read
  + phantom reads
  + Can be allowed or prevented, depending on the ANSI-standard transaction isolation level in use
* Lost updates
* Excessive blocking
* Deadlocks

Pessimistic VS Optimistic

**Pessimistic approach to concurrency;** Assumes that enough concurrent data modification operations are in the system such that **problems will occur, and will lead to data integrity issues unless it takes measures to prevent them**. In a pessimistic concurrency environment, **readers block writers** and **writers block readers.**

**Optimistic concurrency**, by contrast, assumes that there are sufficiently few conflicting data modification operations in the system that **any single transaction is unlikely to modify data that another transaction is modifying**

Under the snapshot-based isolation levels, **SELECT operations do not acquire shared locks**; instead, they simply **read** the required row versions, **consistent with the time the query or transaction started**, **from the version store**, and thus do not block modification operations.

## Transactions

**ACID** principal:

Atomacy: A transaction is treated as a single unit of work.

Consistency: A transaction will leave data in a meaningful state when it completes.

Isolation: The changes that one transaction makes should not interfere with the changes that another transaction makes; each transaction should be executed as if it were the only work that the database system was performing.

Durability: Once a transaction completes, its effects are permanent and recoverable.

### Transaction Scope (beginning and end of an transaction)

**Auto-commit**

The default types of transactions are **auto-commit** transactions and explicit transactions.

An auto-commit transaction is any single data modification operation.

**Explicit**

An **explicit** transaction uses the BEGIN TRANSACTION (or BEGIN TRAN) statement to indicate the beginning of the transaction, and either a COMMIT TRANSACTION or a ROLLBACK TRANSACTION statement to indicate the end.

The non-default types of transactions are **implicit** transactions and **batch-scoped** transactions.

**Implicit**

For **implicit** transactions, a session must be in implicit transaction mode, invoked with a SET option: SET IMPLICIT\_TRANSACTIONS ON. In implicit transaction mode, the start of any transaction is implied. In other words, any data manipulation language (DML) statement (such as INSERT, UPDATE, DELETE and even SELECT) will automatically start a transaction.

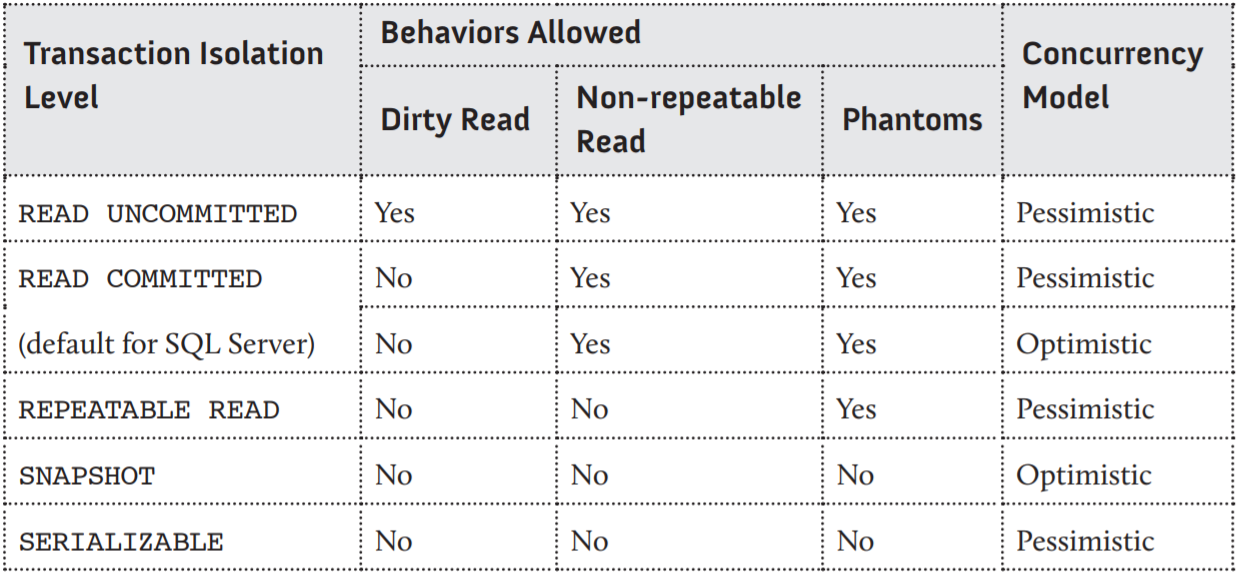
**Batch-scoped**

Introduced in SQL Server 2005, we invoke batch-scoped transactions by requesting the option Multiple Active Result Sets (or MARS) in the client connection string. In those connections, SQL Server will roll back any batch that includes a BEGIN TRAN but does not include a COMMIT TRAN. The purpose of MARS is to avoid a problem called "application deadlock," which we'll discuss in Chapter 4, in the section on sharing locks across connections.

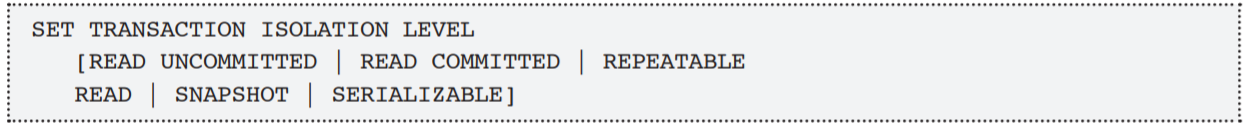
### 

### Transaction isolation

4 levels defined in ANSI Standard:



Can be set using:



### Preventable read phenomena

* Dirty reads
* Non-repeatable reads
* Phantom reads

### READ UNCOMMITTED

READ UNCOMMITTED isolation level allows a transaction to read any data currently on a data or index page, regardless of whether or not the transaction that wrote that data has been committed.

### READ COMMITTED

READ COMMITTED is SQL Server's default isolation level. It ensures that an operation will never read data another transaction has changed but not committed.

So READ COMMITTED behavior has two aspects. Firstly, it prevents dirty reads but, secondly, it still allows non-repeatable reads and phantom reads

### REPEATABLE READ

The REPEATABLE READ isolation level adds to the properties of READ COMMITTED by ensuring that if a transaction re-reads data, or if a query is reissued within the same transaction, then the same data will be returned.

### SERIALIZABLE

if a query is reissued, no data will have changed and no new rows will appear. In other words, we won't see phantoms if the same query is issued twice within a transaction.

### SNAPSHOT

There is an entirely new isolation level, introduced in SQL Server 2005 called SNAPSHOT isolation. The only implementation of snapshot isolation uses optimistic concurrency, so we'll save the discussion of this level until the Chapter 6, on optimistic concurrency.

### Selecting the right isolation level

While preventing blocking, by selecting the READ UNCOMMITTED level, might seem attractive from a concurrency perspective, the price to pay is the prospect of reading incorrect data. At the same time, while preventing all read phenomena, and so guaranteeing more consistent data, is a "good thing," be aware of the tradeoffs in setting your isolation level too high, which is the added cost of acquiring and managing locks, and blocking other processes while those locks are held.

The optimistic isolation levels reduce the amount of blocking, but they do not eliminate it. In addition, they have costs and caveats of their own.

# Chapter 2: Locking Basics

Locking is the activity that occurs when a SQL Server session takes "ownership" of a resource prior to performing a particular action on that resource, such as reading or updating it. Locking is an essential mechanism for allowing multiple users to access and operate on data in a way that avoids inconsistencies in that data.

* The unit of data locked (lock resource) – such as row, page, or table
* The type of locks acquired (lock mode) – shared, exclusive, update, and so on
* The duration of the lock – how long the lock is hel
* Lock ownership – the "scope" of the lock (most locks are transaction scoped)
* Lock metadata – how to review current locking using the Dynamic Management View (DMV) called sys.dm\_tran\_locks.

### 

### Lock resources

When SQL Server locks a row in an index, it refers to it, and displays it, as a KEY lock

Locks on rows in a heap table (one without a clustered index) appear as RID (Row ID) locks.

If the isolation level is SERIALIZABLE, we have a special situation, as SQL Server needs to prevent phantom reads. For this, it employs key-range locks. A key-range lock implies a locked range of index rows including all values greater than the value of the index key that precedes the locked row, and ends with the locked row

## Lock modes

SQL Server uses several types of locks, referred to as lock modes. These include shared locks, exclusive locks, and update locks, used to achieve the four required ANSI modes of transaction isolation.

### Shared locks

By default, SQL Server acquires shared (S) locks automatically when it reads data. A table, page, or individual row of a table or index can hold an S lock. In addition, to support SERIALIZABLE transaction isolation, SQL Server can place S locks on a range of index rows.

### Exclusive locks

SQL Server automatically acquires exclusive (X) locks on data in order to modify that data, during an INSERT, UPDATE, or DELETE operation. Only one transaction at a time can hold an X lock on a particular data resource, and X locks remain until the end of the transaction. The changed data is usually unavailable to any other process until the transaction holding the lock either commits or rolls back.

### Update locks

Update (U) locks are not really a separate kind of lock, but rather a hybrid of S and X locks. A transaction acquires a U lock when SQL Server executes a data modification operation, but first needs to perform a search to find the resource (for example, the row of data) to modify.

SQL Server doesn't need to place an X lock on the row until it is ready to perform the modification, but it does need to apply some sort of lock as it searches, to protect that same data from modification by another transaction in the time between finding the data and modifying it. Therefore, SQL Server places a U lock on the row, checks the row and, if it meets the criteria, converts it to an X lock.

### Intent locks

Intent locks do not represent a distinct mode of locking. The term "intent" is a qualifier to the modes just discussed. In other words, you can have intent shared (IS) locks, intent exclusive locks (IX), and even intent update locks (IU), indicated in the request\_mode column of the sys.dm\_tran\_locks view by IS, IX and IU, respectively.

### Lock duration

The length of time that SQL Server holds a lock depends primarily on the mode of the lock and the transaction isolation level that is in effect.

### Lock ownership

We can think of lock ownership as the scope of the lock, and it can affect lock duration. There are three default values for the lock owner, and two additional types of lock ownership that must be explicitly requested.

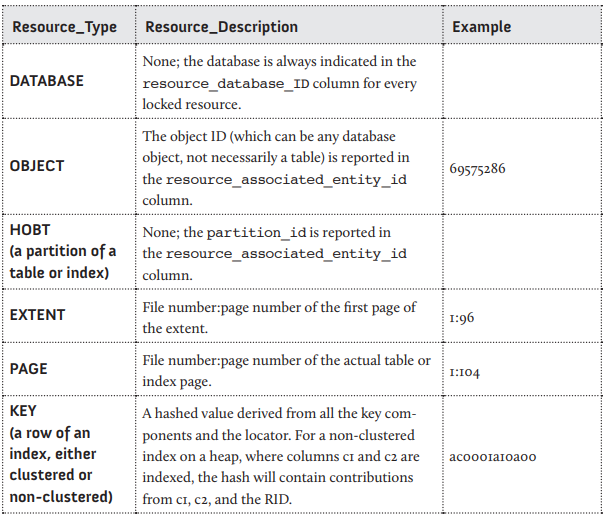
* TRANSACTION – Most of the locks discussed in this book are transaction-owned locks
* SHARED\_TRANSACTION\_WORKSPACE – Every connection in any database (other than master or tempdb) acquires a lock with this owner by.
* EXCLUSIVE\_TRANSACTION\_WORKSPACE – SQL Server acquires a lock with this owner whenever it needs exclusive access to the database.

### Locking metadata

The best source of current lock information is the previously referenced sys.dm\_tran\_ locks DMV. This view replaces the sp\_lock procedure, although sp\_lock is still available. Although calling a procedure might require less typing than querying the sys.

### Resource columns

Six of the columns in sys.dm\_tran\_locks have the resource\_ prefix, and of these resource\_type and resource\_description are probably the most useful, providing the target resource for the requested lock (key, page, and so on) and the identity of the actual resource locked.



### Request columns

There are 13 columns in sys.dm\_tran\_locks used to identify information about the request for the lock, but two of them are documented as being for informational purposes only, not supported. Another two are only useful for DTC transactions or transactions using the MARS protocol, we won't discuss them further. Below is a list of the other nine with a basic explanation of their meaning.

### Summary

In this chapter, we looked at the basics of SQL Server's default locking behavior; the types of locks that SQL Server can acquire, the granularity of the lock, and the duration of the locks. We saw how the locking behavior changes, depending on the transaction isolation level, in order to enforce the behaviors required by the definition of the isolation level. Finally, we looked at multiple examples of locking in various transactions, and examined the locks acquired using SQL Server's lock metadata.