**Manage Isolation Levels**

Isolation Levels:

* Specify how read operations should behave when other concurrent transactions are changing data
* Ensure that queries return complete and consistent results while other concurrent processes are running

How is this achieved?

* Control whether a lock is acquired during a read
* The type of lock is important
* The type of rows being accessed (are they being changed by other transactions or accessing other committed rows?)
* Block transactions where resource requires an exclusive lock

|  |  |
| --- | --- |
| **Lower the Isolation level** | **Increase the Isolation level** |
| Increases the number of concurrent transactions | Minimise concurrency problems |
| Increases the risk of dirty reads | Performance likely to suffer |
|  | Transactions are more likely to block one another |

|  |  |
| --- | --- |
| **Pessimistic Isolation Levels** | **Optimistic Isolation Levels** |
| Use blocking to avoid conflicts | Use snapshots of the data to enable higher concurrency |
| Rely on locks to prevent changes during read operations and to block operations on data being changed by other processes | Make a copy of the data for read operations so write operations can proceed unhindered |
| *Examples* | *Examples* |
| READ COMMITTED | SNAPSHOT |
| SERIALIZABLE | READ COMMITTED SNAPSHOT |

**READ COMMITTED**

* Default Isolation level
* Uses pessimistic locking to protect data
* A transaction cannot read data that is being added or changed by another transaction
* No dirty reads
* Non-repeatable reads or phantom reads are possible
* Transactions under this Isolation level issues shared (S) locks but releases row or page locks after reading a row
* **Note**: If a query scans and index while another transaction changes the index key column of a row, the row in question could be duplicated in the results if the key change moved the row to a new position ahead of the scan.
* SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

**READ UNCOMMITTED**

* Least restrictive setting
* Transactions read uncommitted data
* Ignores locks and reads data from memory
* Allows dirty reads, non-repeatable reads and phantom reads
* Transactions do not acquire shared locks to prevent other transactions from changing data being read
* Transactions set to this level execute quickly because locks and validations are ignored
* You can force the read uncommitted isolation level by using the NOLOCK hint
* **Note**: If a Transaction reads rows using an allocation order scan when another transaction causes a pages split, the query can miss rows.
* **Note2**: Not a good choice for LOB applications where accuracy is critical, but does provide performance benefits
* SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

**REPEATABLE READ**

* Ensures any data read by one transaction is not changed by another transaction
* Much like READ COMMITTED but ensures multiple reads of the same data is consistent
* Dirty reads and non-repeatable reads are prevented
* Phantom reads are possible because range locks are not used
* Prevents changes to existing data but allows the insertion of new data
* Data is protected by shared (S) locks
* **Note**: Only data that has been read is protected. If another transaction inserts a row the first transactions repeat of its query could return a phantom read
* SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

**SERIALIZABLE**

* Most pessimistic
* Uses range locks to prevent changes and insertions
* Behaves like REPEATABLE READ but ensures new rows added after the beginning of the transaction are not visible to the transactions statement
* Dirty reads, non-repeatable reads and phantom reads are prevented
* Blocks write/insert operations
* Phantom reads are not possible
* Each transaction is completely isolated even when executing in parallel or overlap
* SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

**SNAPSHOT**

* Optimistic isolation level
* Allows concurrent read/write operations without blocking
* Database level configuration (using ALTER DATABASE) required then levels set at transaction level
* Provided transaction is open, the state of the committed data at the start of the transaction is preserved, any changes to the data are stored in *tempdb* as well as the transaction sequence number so SQL can determine which version to use for the new transaction’s snapshot
* Increases concurrency by eliminating the need for locks for read operations
* Gives the same data for the duration of the transaction
* Prevents dirty reads, non-repeatable reads and phantom reads
* **Note**: Can’t use with distributed transactions or enable it in *master, tempdb or msdb*
* **Note2:** Enable the ALLOW\_SNAPSHOT\_ISOLATION database option for *tempdb* in order to access global temporary tables with this isolation level
* SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

**READ\_COMMITTED\_SNAPSHOT**

* Optimistic alternative to READ COMMITTED
* Database level configuration required then levels set at transaction level
* Can be used with distributed transactions
* Difference to SNAPSHOT: as each statement executes, SQL takes a snapshot which remains consistent until the next statement executes
* **Note**: Use when application executes long running queries which require data to be consistent at the point the query starts

**Factors to consider:**

* Find the appropriate balance between protecting the data and the effect of each isolation
* Keep transactions short and concise to avoid locking contention and improve overall perfomance