**DEADLOCKS**

* Occurs when 2 processes are competing for exclusive access to a resource
* Essentially a stand off where neither process can proceed
* Resolved by killing one of the processes
* SQL automatically detects deadlocks and selects a process as the victim
* Deadlocks do not only occur on locks, from SQL Server 2012 onward , deadlocks can also happen with memory, MARS (Multiple Active Result Sets) resources, worker threads and resources related to parallel query execution

**Deadlock Types**

* *A Cycle Deadlock*: Occurs when Process A which is holding a lock on Resource X is waiting to obtain an exclusive lock on Resource Y and at the same time Process B is locking Resource X in order to obtain an exclusive lock on Resource X
* *Conversion locks deadlock*: Occurs when a thread tries to convert a lock from one type to another can’t because another thread is also holding a shared lock on the same resource

**Handling Deadlocks**

* Lock manager (LOCK\_MONITOR thread) automatically searches for deadlocks every 5secs
* Chooses a victim and kills terminates the transaction, rolls it back and kills the connection to release all resources
* Deadlock search frequency is automatically adjusted as required (default is 5 secs)
* Victim is chosen based on the deadlock priority setting of a transaction

**Minimise Deadlocks**

* Always try to hold locks for as short a period as possible
* Always access resources in the same order
* Ensure that you don’t have to wait on user input in the middle of a transaction First get all the information you need and then submit the transaction
* Try to limit lock escalation, by using hints such as NOLOCK (on SELECT statements) and HOLDLOCK and UPDLOCK on transaction statements
* Enclose transactions in TRY/CATCH block and add retry logic
* Add a new covering non-clustered index to provide another way for SQL to read data without requiring access to the underlying table (additional overhead)
* Use READ COMMITTED SNAPSHOT or SNAPSHOT which avoid most blocking problems without the risk of dirty reads (both require resources on *tempdb*)

**Monitor Deadlocks**

* Capture and analyse deadlock graphs using SQL Server Profiler or Extended events
* A deadlock graph is an XML description of a deadlock
* Also Trace Flags 1204 and 1222, enabled using DBCC TRACEON(1204,1222,-1)
* When using profiler to capture a deadlock graph, the trace must be configured before the deadlock occurs using the following events:

1. Deadlock graph
2. Lock: Deadlock
3. Lock: Deadlock Chain