What is the solution of concurrency?

**Most operating systems use a combination of preemptive and cooperative multitasking**.

***Preemptive*** means that the system is interruptible when some device or event ask for attention “the OS handles it ASAP”.

***Cooperative multitasking*** on the other hand means the processes running on the system voluntarily yield control of the CPU usually in a time-based fashion.

SQL Server is an example which run cooperatively “under the covers the SQL operation system or SQL OS for short, startup sessions” yielding control at regular intervals.

Other the preemptive side it handles interrupts as they come in, notifying waiting sessions that some unit work is ready.

--------------------------------------------------------------------------------------------------

The relational database system must comply with ACID principle” this is normally implemented through transactions.”

A 🡺 Atomicity ‘all changes take place or none, note: **xact\_abort** in setting if failed roll back transaction ‘

C 🡺 Consistency ‘all data must be in a consistent state.’

I 🡺 Isolation ‘Modification made by one transaction must be isolation from those made by others concurrent transactions.’

D 🡺 Durability ‘When a transaction is complete the results are stored permanently in the system and persist even if a system failure occurs. **in another word** any change happened in transaction logs first’

--------------------------------------------------------------------------------------------------------------------

***Understanding Responsibility***

|  |  |  |
| --- | --- | --- |
|  | SQL Programmers | SQL Server Database Engine |
|  | starting and ending transactions | ensuring the physical integrity of each transaction. |
|  | complete a logical unit of work | locking facilities that preserve transaction isolation. |
|  |  | logging facilities that ensure transaction durability  ‘even if the server hardware, operating system, or the instance of the database engine itself fails; it must be able to rollback any uncompleted work when the system restarts. It uses the transaction log for that.’ |

--------------------------------------------------------------------------------------------------------------------

- By the way, @@VERSION is one of many system configuration functions that you can use to see how the system is set up.

- SQL Server runs in auto-commit transaction mode. In this mode, each individual statement is a transaction [begin the transaction the run query then commit the transaction and if any error occurs the transaction is rolled back].

- ROLLBACK says to undo any changes made, ROLLBACK is usually used as part of error handling in complex transactions. When nesting transactions, this single statement rolls back all inner transactions, all the way to the outermost BEGIN TRANSACTION, and then sets the transaction counter to 0. This is the standard behavior of ROLLBACK, though there are some nuances we need to look at.

--------------------------------------------------------------------------------------------------------------------

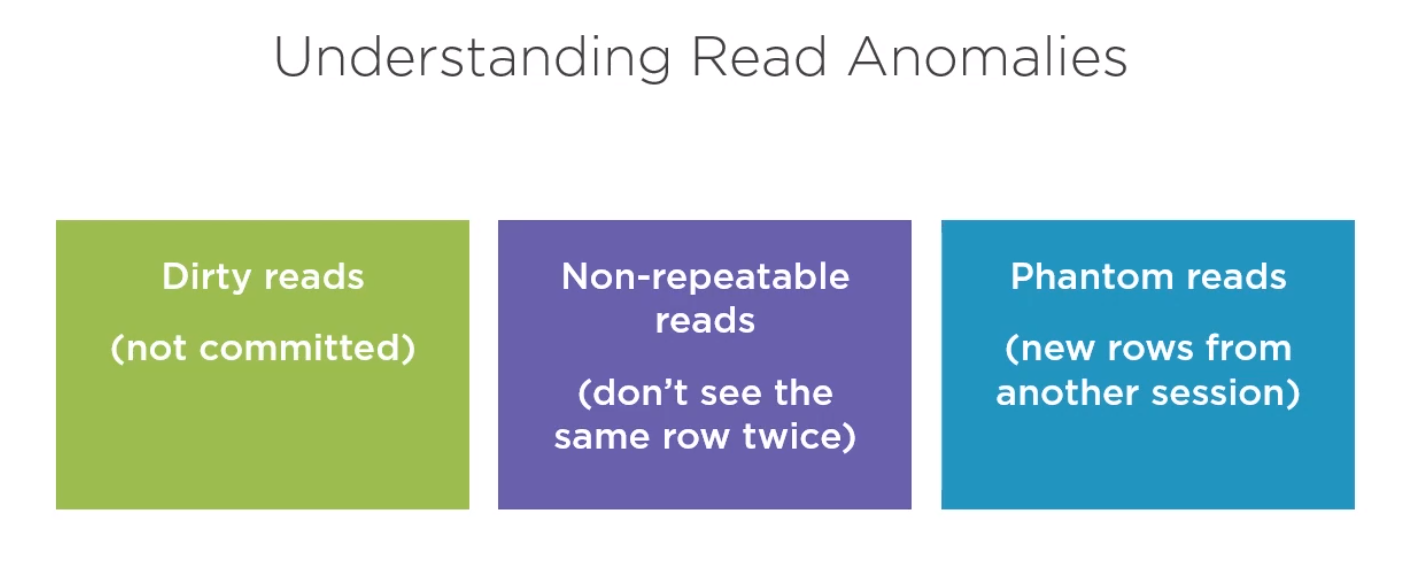
***Transaction Types:***

Implicit transaction and Explicit transaction

The SAVE statement sets a savepoint when ROLLBACK statement is used for this operation and specifies a previously set savepoint to return to.

--------------------------------------------------------------------------------------------------------------------

***Isolation Levels***



We have special kind of lock called **Schema Stability** lock when running under READ UNCOMMITTED. This lock blocks any changes to the DDL operations and some DML that might change the table definition while it is being read.

**Read Uncommitted** [anything goes. At any moment (**No Lock**)]

*you can get all three of the major anomalies, dirty reads, nonrepeatable reads, and phantom reads.*

The default isolation for most transactions is **READ COMMITTED** *[no dirty reads].*

**REPEATABLE READ.**

**SERIALIZABLE.**

***Snapshot Isolation***

**Snapshot isolation which uses optimistic concurrency.**

**🡺 The SQL server database engine maintains versions of each row that is modified and when data is read it uses a version of the data that was committed at the time the reading transaction began.**

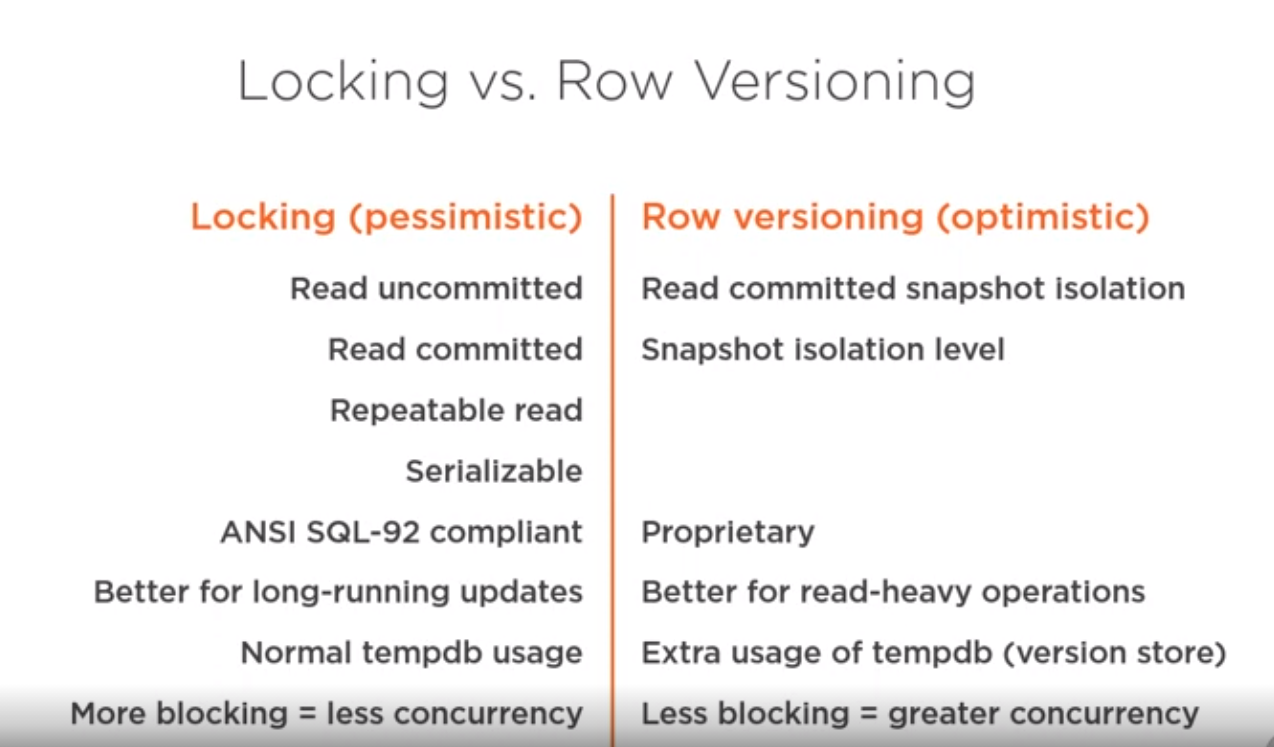
**🡺 the Chance that read operation will block other transactions is greatly reduced.**

**🡺 SQL server uses a copy-on-write mechanism when a row is modified or deleted.**

**🡺 tempdb is used to hold the version store.**

**Dynamic Management Views (DMVs) that show information about active snapshot transactions.**

***Locking and Row versioning***



**Note: The first difference is the four-level isolation level are compliant with ANSI SQL- 92 standard and SQL row versioning using snapshot isolation is proprietary**

***Locking in the SQL sever engine.***

The locks are about the database,page,key.object.

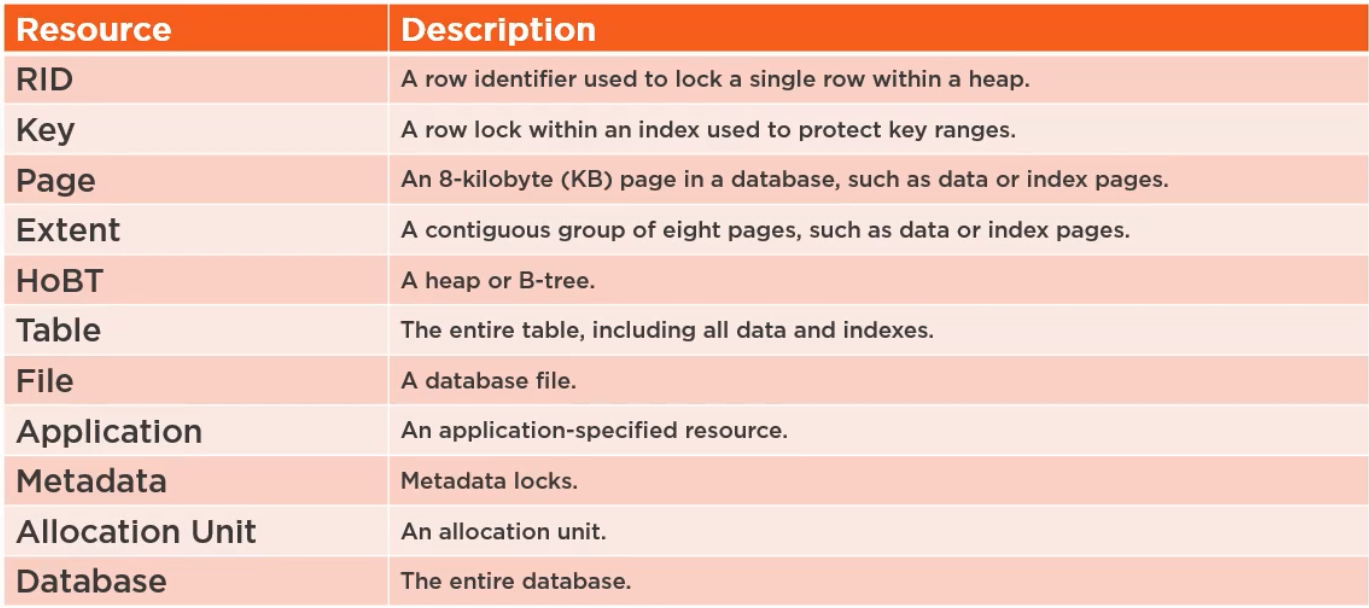
Exec dbo.Sp\_WhoIsActive; => is a free PROC it tell you what’s going on to show sessions and also shows blocking sessions

Sql server uses multigranular locking. This means that different types of resources can be locked by a transaction

**The sys.dm.tanc.locks Dynamic Management View (DMVs)will show us the state of affairs.**

**This is a perfect, if simple, picture of a locking hierarchy. We can see the different granularities from database down to page and the different requested modes and status of each lock. If you have an active SQL Server instance that you work with, take a moment and see what you can find out using this DMVs.**

***• Hierarchy of Locking granularities (from the lowest to the highest):***



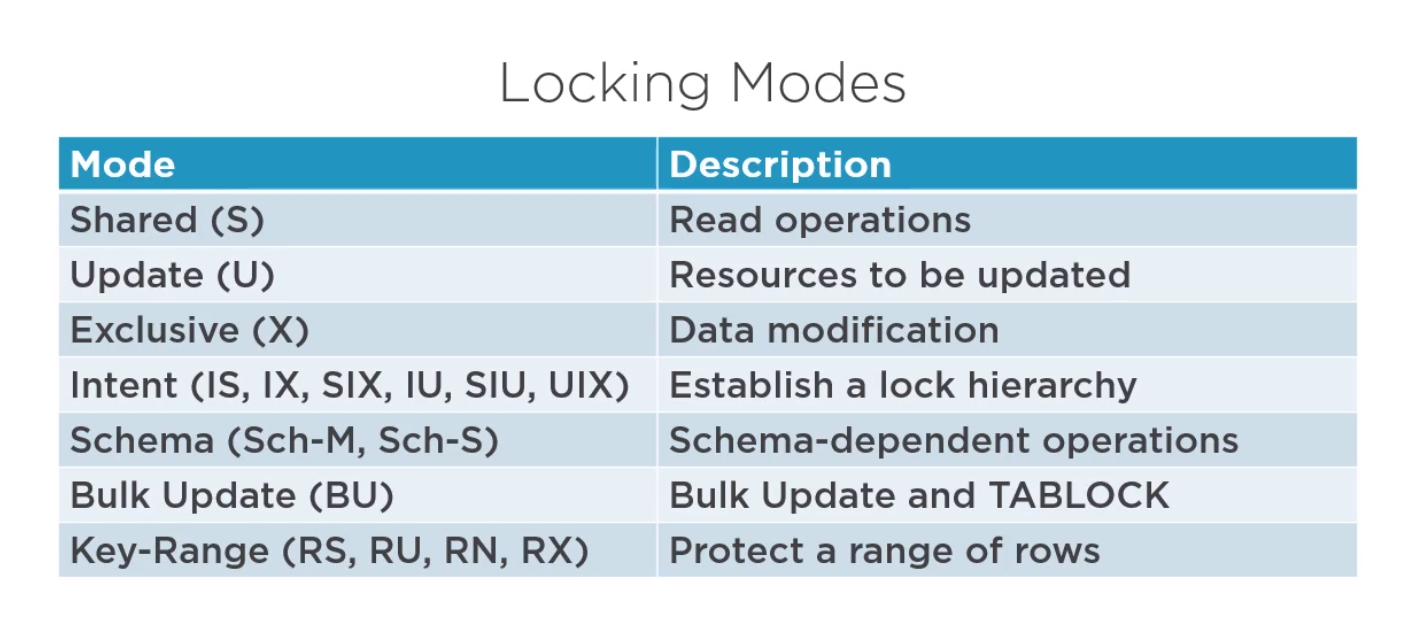
1. **RID:** which is the lowest granularity level, is the RID or row ID used to lock a single row within a heap.

Recall that a table is either a heap or a clustered index.

1. **Key:** is used to lock a row within an index, which, of course, includes clustered indexes. It is used to protect a range of rows in a SERIALIZABLE transaction. It does this by preventing the insertion of new rows with keys that fall within the range of keys being read by the transaction. In this way, phantom reads are prevented.
2. **Page**: is the next higher level. This represents an 8 KB page in the database, and could be a data or index page.
3. **Extent:** this happens on a table or index needs to grow and a new extent is required.
4. **HoBT:** Heap or B-Tree Lock: is a lock that is protecting B-Tree (that’s an index), a heap data page is in a table that does not have a clustered index.
5. **File Database:** in simple databases, there might be 2 files but in complex ones, there are thousands of files. Any operation in one of those files needs a file log.
6. **Application:** Such as .Net Apps.
7. **Metadata:** is a lock on a piece of catalog information.
8. **Allocation Unit:** is used to store all the data and indexes belonging to a table partition (all tables have at least on partition usually called primary partition and may have many more depending on the design). These locks might appear when you drop or rebuild large indexes or tables since freeing those unused pages is not done until the transaction is completed.
9. **Database** Lock locks the whole Database.

***Locking Modes***

There are 7 modes used by SQL server some of which have more than one sublevel



Shared locks allow concurrent transcation to ready using **pessimistic concurrency** . they prevent modifications by other transcations as long as the shared lock is held . This lock mode is not used with snapshot isolation since it would be redundant.

---------------------------------------------------------

Update locks are used to prevent common deadlocks.Only one transcation can hold an update lock on a given resource which can prevent deadlock situations.

-----------------------------------------------------------

Exclusive locks prevent access to the locked resouces by other transcations.Also, read operations will only work on those resouces when using the **READ UNCOMMITTED OR NOLOCK Isolation level.**

Since inseat, update and delete first read data before changing it”those operation usually request both shared and execlusive logs”

-----------------------------------------------------------------------

Intent locks are special and have 6 sublevels , they are called intent because they are acquired before a full lock and show the intention to place locks at that level.

Intent locks also help the Database Engine to detect lock conflicts at higher levels in the hierarchy, which improves efficiency.

--------------------------------------------------------------------------

at this point concurrency and lock granularity have an inverse relationship

. At the lowest level, row or key locks can deliver maximum concurrency.

***Lock Escalation:***

**Dynamic Locking:** to keep a balanced relationship between concurrency and Granularity for example a range lock for some rows can change to be table lock depending on the number of the rows. This process called **Lock Escalation. We can set this per table by Specifying its type:**

* Auto: SQL serve can handle this automatically.
* Table: Always set this per table.
* Disable: Disable the feature completely.

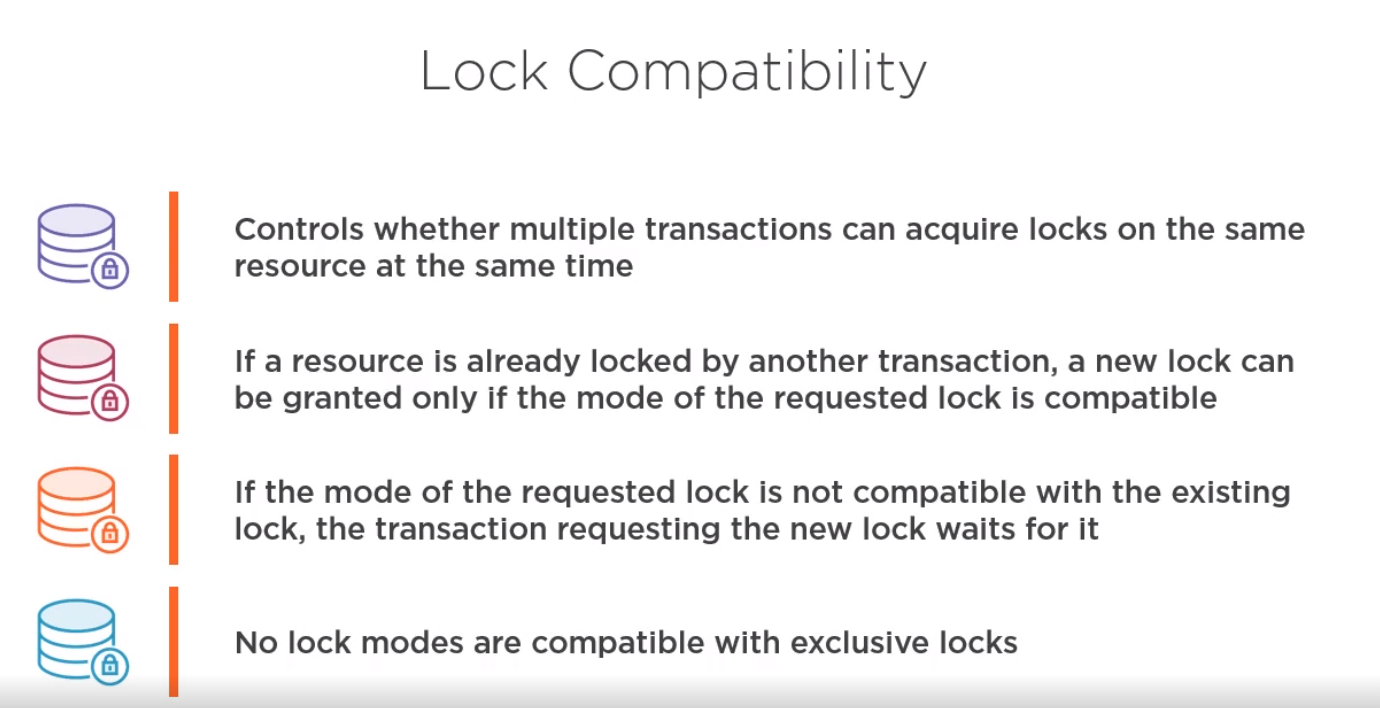
-Pros of Lock Escalation:

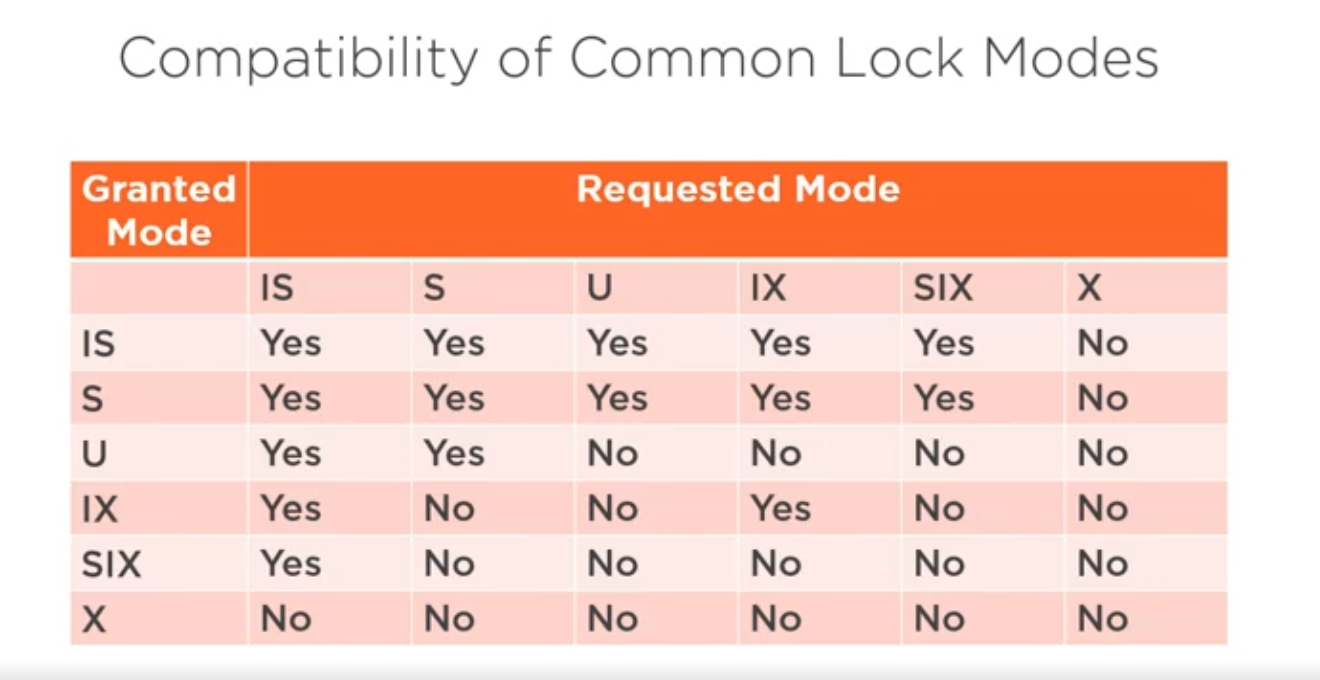
* Simplifies Database Administration. Because the SQL server handles this automatically.
* Increases Performance.
* Automatic Lock adjustment.

you can indicate what type of escalation behavior you want to use at the table level. You can choose between AUTO, which is the process I just described, TABLE, which forces escalation to the table level, even when a lower granularity is available, or DISABLE, which prevents lock escalation, in most cases. But note that this control is given to handle special cases. Normally, you should let SQL Server manage locking dynamically, and only use this option when that appears to be suboptimal for some case. Also, before you use controls like this one, benchmark your current performance, then check for improvements or regressions.

Side note :Two sessions cannot have exclusive locks on the same object at the same time,

**Locking is** the strategy used to maintain integrity in an active database with many sessions querying and updating data.



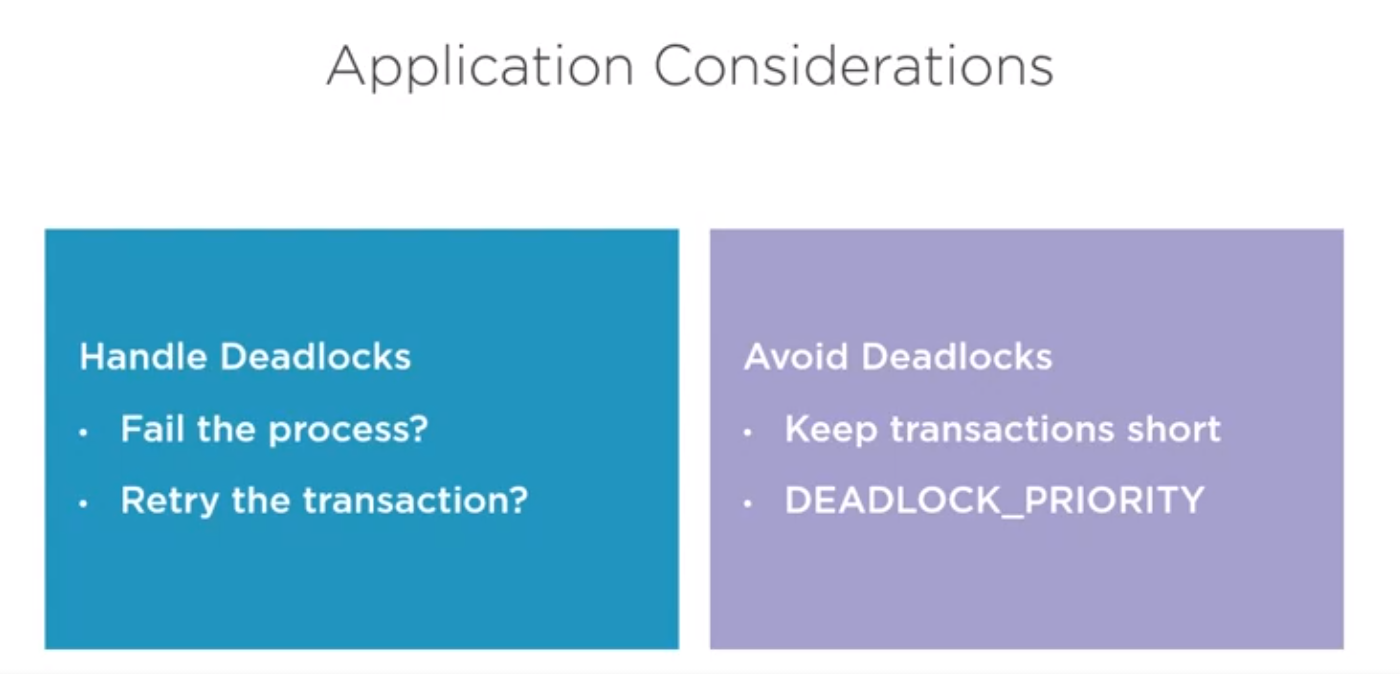


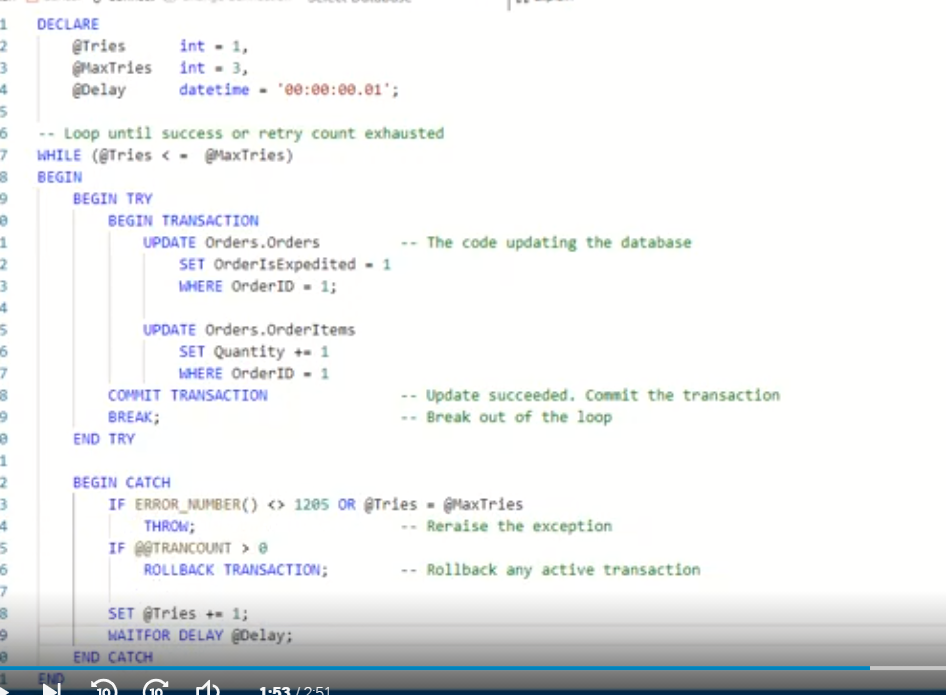
How SQL knows the DeadLock was happening?

SQL server has a deadlock detector that runs by default every 5 seconds, when deadlock is detected it has to do something about it .it look at the transcations that are deadlocked and chooses one of them to live and onther to die.

We have avery useful session called system\_health events => filter it to only looks to deadlocks and you will find there is a report of deadlock that just happened if you selected it and click deadlock tab it shows you a nice graph showing the situation

-- two advanced feature in sql server : Extened events and XML - based querying using XQuery.





<https://gist.github.com/gbritton1/a3ccb4333ba2dcce59fd0abf002a0e31>