**In SQL Server, a database query can execute as a logical unit of work in itself, or it can participate in a biggerlogical unit of work. A bigger logical unit of work can be defined using the BEGIN TRANSACTION statement alongwith COMMIT and/or ROLLBACK statements. Every logical unit of work must conform to a set of four properties called *ACID* properties:**

**Atomicity**

**Consistency**

**Isolation**

**Durability**

**Atomicity**

**A logical unit of work must be *atomic.* That is, either all the actions of the logical unit of work are completed or no effect is retained. To understand the atomicity of a logical unit of work, consider the following example**

**--Create a test table**

**IF (SELECT OBJECT\_ID('dbo.ProductTest')**

**) IS NOT NULL**

**DROP TABLE dbo.ProductTest ;**

**GO**

**CREATE TABLE dbo.ProductTest**

**(ProductID INT CONSTRAINT ValueEqualsOne CHECK (ProductID = 1)**

**) ;**

**GO**

**--All ProductIDs are added into t1 as a logical unit of work**

**INSERT INTO dbo.ProductTest**

**SELECT p.ProductID**

**FROM Production.Product AS p ;**

**GO**

**SELECT \***

**FROM dbo.ProductTest ; --Returns 0 rows**

**UPDATE**

**SQL Server treats the preceding INSERT statement as a logical unit of work. The CHECK constraint on columnc1 of thedbo.t1 tableallows only the value of 1. Although the ProductID column in the Production.Producttable starts with the value of 1, it also contains other values. For this reason, the INSERT statement won’t add**

**any records at all to the dbo.t1 table, and an error is raised because of the CHECK constraint. This atomicity isautomatically ensured by SQL Server.**

**So far, so good. But in the case of a bigger logical unit of work, you should be aware of an interesting behaviorof SQL Server. Imagine that the previous insert task consists of multiple INSERT statements. These can becombined to form a bigger logical unit of work, as follows:**

**BEGIN TRAN**

**--Start: Logical unit of work**

**--First:**

**INSERT INTO dbo.ProductTest**

**SELECT p.ProductID**

**FROM Production.Product AS p ;**

**--Second:**

**INSERT INTO dbo.ProductTest**

**VALUES (1);**

**COMMIT --End: Logical unit of work**

**GO**

**With the dbo.ProductTest tablealready created in the --atomicity script, the BEGIN TRAN and COMMIT pairof statements defines a logical unit of work, suggesting that all the statements within the transaction shouldbe atomic in nature. However, the default behavior of SQL Server doesn’t ensure that the failure of one of thestatements within a user-defined transaction scope will undo the effect of the prior statement(s). In the precedingtransaction, the first INSERT statement will fail as explained earlier, whereas the second INSERT is perfectly fine.**

**The defaultbehavior of SQL Server allows the second INSERT statement to execute, even though the first INSERTstatement fails. A SELECT statement, as shown in the following code, will return the row inserted by the secondINSERT statement:**

**SELECT \*FROM dbo.ProductTest ; --Returns a row with t1.c1 = 1**

**The atomicity of a user-defined transaction can be ensured in the following two ways:**

**• SET XACT\_ABORT ON**

**• Explicit rollback**

**Let’s look at these quickly.**

SETXACT\_ABORTON

**You can modify the atomicity of the INSERT task in the preceding section using the SET XACT\_ ABORT ONstatement:**

**SET XACT\_ABORT ON**

**GO**

**BEGIN TRAN**

**--Start: Logical unit of work**

**--First:**

**INSERT INTO dbo.ProductTest**

**SELECT p.ProductID**

**FROM Production.Product AS p**

**--Second:**

**INSERT INTO dbo.ProductTest**

**VALUES (1)**

**COMMIT**

**--End: Logical unit of work GO**

**SET XACT\_ABORT OFF**

**GO**

**The SET XACTABORT statement specifies whether SQL Server should automatically roll back and abort anentire transaction when a statement within the transaction fails. The failure of the first INSERT statement willautomatically suspend the entire transaction, and thus the second INSERT statement will not be executed.**

**The effect of SET XACTABORT is at the connection level, and it remains applicable until it is reconfigured or theconnection is closed. By default, SET XACT\_ABORT is OFF.**

**Explicit Rollback**

**You can also manage the atomicity of a user-defined transaction by using the TRY/CATCH error-trappingmechanism within SQL Server. If a statement within the TRY block of code generates an error, then the CATCHblock of code will handle the error. If an error occurs and the CATCH block is activated, then the entire work of a**

**user-defined transaction can be rolled back, and further statements can beprevented from execution, as follows.**

**BEGIN TRY**

**BEGIN TRAN --Start: Logical unit of work --First:**

**INSERT INTO dbo.t1**

**SELECT p.ProductID**

**FROM Production.Product AS p**

**Second:**

**INSERT INTO dbo.t1**

**VALUES (1)**

**COMMIT --End: Logical unit of work**

**END TRY**

**BEGIN CATCH**

**ROLLBACK**

**PRINT 'An error occurred'**

**RETURN**

**END CATCH**

**The ROLLBACK statement rolls back all the actions performed in the transaction until that point.Since the atomicity property requires that either all the actions of a logical unit of work are completed orno effects are retained, SQL Server isolates the work of a transaction from that of others by granting it exclusiverights on the affected resources. This means that the transaction can safely roll back the effect of all its actions,if required. The exclusive rights granted to a transaction on the affected resources block all other transactions(or database requests) trying to access those resources during that time period. Therefore, although atomicity is**

**required to maintain the integrity of data, it introduces the undesirable side effect of blocking.**

**Consistency**

**A logical unit of work should cause the state of the database to travel from one consistent state to another.At the end of a transaction, the state of the database should be fully consistent. SQL Server always ensures thatthe internal state of the databases is correct and valid by automatically applying all the constraints of the affecteddatabase resources as part of the transaction. SQL Server ensures that the state of internal structures, such as dataand index layout, are correct after the transaction. For instance, when the data of a table is modified, SQL Serverautomatically identifies all the indexes, constraints, and other dependent objects on the table and applies thenecessary modifications to all the dependent database objects as part of the transaction.**

**The logical consistency of the data required by the business rules should be ensured by a databasedeveloper. A business rule may require changes to be applied on multiple tables. The database developer shouldaccordingly define a logical unit of work to ensure that all the criteria of the business rules are taken care of. SQL**

**Server provides different transaction management features that the database developer can use to ensure thelogical consistency of the data.**

**Therefore, even though consistency is required to maintain a valid logical and physical state of thedatabase, it also introduces the undesirable side effect of blocking.**

**Isolation**

**In a multiuser environment, more than one transaction can be executed simultaneously. These concurrenttransactions should be isolated from one another, so that the intermediate changes made by one transactiondon’t affect the data consistency of other transactions. The degree of isolation required by a transaction can vary.SQL Server provides different transaction isolation features to implement the degree of isolation required by atransaction.**

**The isolation requirements of a transaction operating on a database resource can block other transactionstrying to access the resource. In a multiuser database environment, multiple transactions are usually executedsimultaneously. It is imperative that the data modifications made by an ongoing transaction be protectedfrom the modifications made by other transactions. For instance, suppose a transaction is in the middle ofmodifying a few rows in a table. During that period, to maintain database consistency, you must ensure that othertransactions do not modify or delete the same rows. SQL Server logically isolates the activities of a transactionfrom that of others by blocking them appropriately, which allows multiple transactions to execute simultaneouslywithout corrupting one another’s work.**

**Excessive blocking caused by isolation can adversely affect the scalability of a database application.A transaction may inadvertently block other transactions for a long period of time, thereby hurting databaseconcurrency. Since SQL Server manages isolation using locks, it is important to understand the locking**

**architecture of SQL Server. This helps you analyze a blocking scenario and implement resolutions.**

**Durability**

**Once a transaction is completed, the changes made by the transaction should be durable. Even if the electricalpower to the machine is tripped off immediately after the transaction is completed, the effect of all actions withinthe transaction should be retained. SQL Server ensures durability by keeping track of all pre- and post-images ofthe data under modification in a transaction log as the changes are made. Immediately after the completion ofa transaction, SQL Server ensures that all the changes made by the transaction are retained even if SQL Server,the operating system, or the hardware fails (excluding the log disk). During restart, SQL Server runs its databaserecovery feature, which identifies the pending changes from the transaction log for completed transactions andapplies them tothe database resources. This database feature is called roll forward.**

**The recovery interval period depends on the number of pending changes that need to be applied to thedatabase resources during restart. To reduce the recovery interval period, SQL Server intermittently appliesthe intermediate changes made by the running transactions as configured by the recovery interval option.**

**The recovery interval option can be configured using the spconfigure statement. The process of intermittentlyapplying the intermediate changes is referred to as the checkpoint process. During restart, the recovery processidentifies all uncommitted changes and removes them from the database resources by using the pre-images of**

**the data from the transaction log.The durability property isn’t a direct cause of blocking since it doesn’t require the actions of a transactionto be isolated from those of others. But in an indirect way, it increases the duration of the blocking. Since the**

**durability property requires saving the pre- and post-images of the data under modification to the transaction logon disk, it increases the duration of the transaction and blocking.**

**Locks**

**When a session executes a query, SQL Server determines the database resources that need to be accessed; and,if required, the lock manager grants database locks to the session. The query is blocked if another session hasalready been granted the locks; however, to provide both transaction isolation and concurrency, SQL Server uses**

**different lock granularities.**

**Lock Granularity**

**SQL Server databases are maintained as files on the physical disk. In the case of a non- database file such as anExcel file, the file may be written to by only one user at a time. Any attempt to write to the file by other users fails.However, unlike the limited concurrency on a non-database file, SQL Server allows multiple users to modify (oraccess) contents simultaneously, as long as they don’t affect one another’s data consistency. This decreasesblocking and improves concurrency among the transactions.**

**To improve concurrency, SQL Server implements lock granularities at the following resource levels and inthis order:**

**• Row (RID)**

**• Key (KEY)**

**• Page (PAG)**

**• Extent (EXT)**

**• Heap or B-tree (HoBT)**

**• Table (TAB)**

**• File (FIL)**

**• Application (APP)**

**• MetaData (MDT)**

**• Allocation Unit (AU)**

**• Database (DB)**

**Row-Level Lock**

**This lock is maintained on a single row within a table and is the lowest level of lock on a database table. Whena query modifies a row in a table, an RID lock is granted to the query on the row. For example, consider thetransaction on the following test table :**

**--rowlock**

**--Create a test table**

**IF (SELECT OBJECT\_ID('dbo.Test1')**

**) IS NOT NULL**

**DROP TABLE dbo.Test1 ;**

**GO**

**CREATE TABLE dbo.Test1 (C1 INT) ;**

**INSERT INTO dbo.Test1**

**VALUES (1) ;**

**GO**

**BEGIN TRAN**

**DELETE dbo.Test1**

**WHERE C1 = 1 ;**

**SELECT dtl.request\_session\_id,**

**dtl.resource\_database\_id,**

**dtl.resource\_associated\_entity\_id,**

**dtl.resource\_type,**

**dtl.resource\_description,**

**dtl.request\_mode,**

**dtl.request\_status**

**FROM sys.dm\_tran\_locks AS dtl**

**WHERE dtl.request\_session\_id = SPID;**

**ROLLBACK**

**The dynamic management view,sys.dm\_tran\_locks, can be used to display the lock status. The queryagainst sys.dm\_tran**\_**locks shows that the DELETE statement acquired an exclusive RID lock on therow to be deleted.**

**Granting an RID lock to the DELETE statement prevents other transactions from accessing the row.The resource locked by the RID lock can be represented in the following format from the *resource\_description* column:**

***DatabaseID:FileID:PageID:Slot(row)***

**In the output from the query against sys.dm\_tran\_locks, the *DatabaselD* is displayedseparately under the *resource\_database\_id* column. The *resource\_description* column value for the RID typerepresents the remaining part of the RID resource as 1:23593:0. In this case, a FilelD of 1 is the primary datafile, a PagelD of 23593 is a page belonging tothe dbo.Test1 tableidentified by the *Obi\_Id* column, and a Slot**

**(row) of 0 represents the row position within the page. You can obtain the table name and the database name byexecuting the following SQL statements:**

**SELECT OBJECT\_NAME(176719682),**

**DB\_NAME(9) ;**

**The row-level lock provides very high concurrency since blocking is restricted to the row under effect.**

**Key-Level Lock**

**This is a row lock within an index, and it is identified as a KEY lock. As you know, for a table with a clustered index,the data pages of the table and the leaf pages of the clustered index are the same. Since both of the rows are thesame for a table with a clustered index, only a KEY lock is acquired on the clustered index row, or limited range ofrows, while accessing the row(s) from the table (or the clustered index). For example, consider having a clusteredindex on the Test1:**

**CREATE CLUSTERED INDEX TestIndex ON dbo.Test1(C1) ;**

**Next, rerun the following code:**

**BEGIN TRAN**

**DELETE dbo.Test1**

**WHERE C1 = 1 ;**

**SELECT dtl.request\_session\_id,**

**dtl.resource\_database\_id,**

**dtl.resource\_associated\_entity\_id,**

**dtl.resource\_type,**

**dtl.resource\_description,**

**dtl.request\_mode,**

**dtl.request\_status**

**FROM sys.dm\_tran\_locks AS dtl**

**WHERE dtl.request\_session\_id = SPID ;**

**ROLLBACK**

**The corresponding output from sys.dm\_tran\_lockswill show a KEY lock instead of the RID lock,.When you are querying sys.dm\_tran\_locks,you will be able to retrieve the database identifier, resource\_database\_id. You can also get information about what is being locked fromresource\_associated\_entity\_id;however, to get to the particular resource (in this case, the page on the key), you have to go to the resource\_**

**description column for the value, which is (1:23316). In this case, the Indld of 1 is the clustered index onthedbo.Test1 table. You also see the types of requests that are made: S, Sch-S, X, and so on.**

**Like the row-level lock, the key-level lock provides very high concurrency.**

**Page-Level Lock**

**A page-level lock is maintained on a single page within a table or an index, and it is identified as a PAG lock. Whena query requests multiple rows within a page, the consistency of all the requested rows can be maintained by**

**acquiring either RID/KEY locks on the individual rows or a PAG lock on the entire page. From the query plan, thelock manager determines the resource pressure of acquiring multiple RID/KEY locks; and if the pressure is foundto be high, the lock manager requests a PAG lock instead.**

**The resource locked by the PAG lock may be represented in the following format in the resource\_description column of sys.dm\_tran\_locks:**

**DatabaseID:FileID:PageID**

**The page-level lock increases the performance of an individual query by reducing its locking overhead, but ithurts the concurrency of the database by blocking access to all the rows in the page.**

**Extent-Level Lock**

**An extent-level lockis maintained on an extent (a group of eight contiguous data or index pages), and it isidentified as an EXT lock. This lock is used, for example, when an ALTER INDEX REBUILD command is executed ona table and the pages of the table may be moved from an existing extent to a new extent. During this period, theintegrity of the extents is protected using EXT locks.**

**Heap or B-tree Lock**

**A heap or B-tree lock is used to describe when a lock to either type of object could be made. The target objectcould be an unordered heap table, a table without a clustered index, or a B-tree object, usually referring topartitions. A setting within the ALTER TABLE function allows you to exercise a level of control over how locking**

**escalation is affected with the partitions. Because partitions are storedacross multiple filegroups, each one has to have its own data allocation definition. This is where the HoBT comesinto play. It acts like a table-level lock, but on a partition instead of on the table itself.**

**Table-Level Lock**

**This is the highest level of lock on a table, and it is identified as a TAB lock. A table-level lock on a table reservesaccess to the complete table and all its indexes.**

**When a query is executed, the lock manager automatically determines the locking overhead of acquiringmultiple locks at the lower levels. If the resource pressure of acquiring locks at the row level or the page level isdetermined to be high, then the lock manager directly acquires a table-level lock for the query.**

**The resource locked by the TAB lock will be represented in resourcedescription in the following format:**

**DatabaselD:ObjectID**

**A table-level lock requires the least overhead compared to the other locks and thus improves theperformance of the individual query. On the other hand, since the table-level lock blocks all write requests on theentire table (including indexes), it can significantly hurt database concurrency.**

**Sometimes an application feature may benefit from using a specific lock level for a table referred to in aquery. For instance, if an administrative query is executed during nonpeak hours, then a table-level lock may notimpact the users of the system too much; however, it can reduce the locking overhead of the query and therebyimprove its performance. In such cases, a query developer may override the lock manager’s lock level selectionfor a table referred to in the query by using locking hints:**

**SELECT \* FROM <TableName> WITH(TABLOCK)**

**Database-Level Lock**

**A database-level lock is maintained on a database and is identified as a DB lock. When an application makes adatabase connection, the lock manager assigns a database-level shared lock to the corresponding SPID. Thisprevents a user from accidentally dropping or restoring the database while other users are connected to it.SQL Server ensures that the locks requested at one level respect the locks granted at other levels. Forinstance, oncea user acquires a row-level lock on a table row, another user can’t acquire a lock at any other levelthat may affect the integrity of the row. The second user may acquire a row-level lock on other rows or a page level**

**lock on other pages, but an incompatible page- or table-level lock containing the row won’t be granted toother users.**

**The level at which locks should be applied need not be specified by a user or database administrator; thelock manager determines that automatically. It generally prefers row-level and key-level locks when accessing asmall number of rows to aid concurrency. However, if the locking overhead of multiple low-level locks turns outto be very high, the lock manager automatically selects an appropriate higher-level lock.**