28. Write a short essay talking about your understanding of transactions, locks and isolation levels.

ACID test consists of 4 requirements that every transaction must pass successfully:

* Atomicity – requires that a transaction that involves two or more discrete parts of information must commit all parts or none
* Consistency – requires that a transaction must create a valid state of new data, or it must roll back all data to the state that existed before the transaction was executed
* Isolation – requires that a transaction that is still running and did not commit all data yet, must stay isolated from all other transactions
* Durability – requires that committed data must be stored using method that will preserve all data in correct state and available to a user, even in case of a failure

In transaction, the level of isolation should be considered which allows a transaction to execute as if there are no other concurrently running transactions. This is to prevent reads and writes of temporary, aborted, or otherwise incorrect data written by concurrent transactions. But there might have performance cost when the transaction is isolated: transaction latency or throughput. So isolation levels can provide the user of a system the ability to trade off isolation guarantees for improved performance.

There are some anomalies in concurrent systems: dirty read, lost updates, non-repeatable read and phantom read. Accordingly, the isolation levels (read uncommitted, read committed, repeatable read, serializable and snapshot) are defined by which of those anomalies are possible.

Using locks to make sure the isolation levels are achieved. Lock mode considers various lock types that can be applied to a resource that has to be locked:

Exclusive (X)

Shared (S)

Update (U)

Intent (I)

Schema (Sch)

Bulk update (BU)

29. Write a short essay, plus screenshots talking about performance tuning in SQL Server. Must include Tuning Advisor, Extended Events, DMV, Logs and Execution Plan.

Performance tuning is the process of improving SQL queries to accelerate the system performance. The purpose is to reduce the amount of time it takes a user to receive a result after issuing a query, and to reduce the amount of resources used to process a query. Some will affect the query runtime:

* Table size: If the query hits one or more tables with millions of rows or more, it could affect performance.
* Joins: If the query joins two tables in a way that substantially increases the row count of the result set, the query is likely to be slow.
* Aggregations: Combining multiple rows to produce a result requires more computation than simply retrieving those rows.

We can identify slow queries by:

1. Generating an execution plan. Actual execution plans are generated after the queries run.
2. Monitoring resource usage. Using System Monitor allows us to monitor Windows and SQL Server counters simultaneously, so we can verify if there’s any correlation between the performance of the two.
3. Using the database engine tuning advisor. It’s importance to analyze the performance of Transact-SQL statements that are run against the database that is intended to tune. We can use the database engine tuning advisor to analyze the performance implications. For example, it may advise us to create or remove indexes.
4. Finding slow queries with DMVs. Those dynamic management views (DMVs) are built in. There are dozens of them and they can provide a wealth of information about a wide range of topics. There are several DMVs that provide data about query stats (like the screenshot I showed), execution plans, recent queries and much more. For example, the query below can be used to find the queries that use the most reads, writes, worker time (CPU), etc.
5. SQL Server Extended Events. It was a very useful tool to see in real-time what SQL queries are being executed against the database.

(Next page for screenshots)









