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

A transaction is a unit of work that can involve multiple activities such as querying or modifying data. Transactions have four basic properties: atomicity, consistency, isolation, and durability.

Transaction:

Atomicity means a transaction acts as a unit of work, which means either all statements within a transaction commit or none of them do. If the system fails during a transaction, when restarting, system would undo the uncommitted changes which are not recorded in the transaction log. When using error handling, if the error is not severe enough such as primary key constraint violation, then the transaction will not be automatically rollbacked.

Consistency means when multiple concurrent transactions happen, the system needs to make sure the data is consistent, which means after the transactions, the database must be left in a state where all constraints are still satisfied such as domain constraints, key constraints, etc. Each concurrent transaction executed in isolation must keep the database in a consistent state. The level of consistency is depended on the isolation level. The higher the isolation level is, the more consistent the data is.

Isolation: Isolation makes sure each transaction only accesses the consistent data. With each transaction executed in isolation, we can guarantee to have consistent data. The isolation level determines the level of consistency. There are two models for handling isolation: locking mode and row versioning. In locking mode, there are four isolation level, which are READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE. We also have NOLOCK mode. When reading the data in NOLOCK mode, we don’t need to define a shared lock or exclusive lock. This means we may read uncommitted data, which is equivalent to READ UNCOMMITTED. In row versioning, we have SNAPSHOT and READ COMMITTED SNAPSHOT.

Durability: When changes to the database are committed, the commit instructions will be recorded in the transaction log. And this happens even before the changes are actually applied to the data portion on the disk. If the commit instructions of the changes are recorded in the transaction log, then we say the transaction is durable. When system restarts, it will go through the transaction log and do recovery process. In the recovery process, it either perform redo or udo. The redo phase is applied when the changes are recorded in the transaction log but haven’t been applied to data portion yet, and since the transaction is durable, the system will apply these changes. The undo phase means when the changes are not recorded in the transaction log, then the system rollbacks these changes from the transaction.

Locks:

There are mainly three locks: shared lock S, exclusive lock X, and update lock U. When shared lock S is imposed on the data resource, the data resource is reserved to read only. Other transactions cannot modify the data, but can read the same data simultaneously. When exclusive lock X is imposed on the data resource, the data resource will be reserved exclusively, which means other transactions cannot read the same data resource or update it. Update lock U can be imposed on the data resource that already has a shared lock, and in this case the update lock acts as a shared lock, and when the data is ready for the update lock, it will become an exclusive lock, reserving the data resource exclusively. The update lock can impose on data resource that has a shared lock, but a shared lock cannot impose on data resource that has an update lock.

Isolation levels:

As mentioned above, there are two different models to handle isolation. The locking model, also referred to as the pessimistic concurrency control, has four isolation level. The least strict one is READ UNCOMMITTED. In READ UNCOMMITTED, when reader wants to read the data, it does not need to request for a shared lock, and therefore reader has no interference with the writer who modifies the data. If the reader and write executes transactions on the same data resource, then the reader can read data changes that are not committed by the writer. This refers to dirty reads.

In READ COMMITTED, when reader wants to read the data, it has to request for a shared lock. If the writer is granted with a exclusive lock and modifying the same data resource that the reader wants to read, then the reader must wait until the writer releases the exclusive lock. The shared lock in READ COMMITTED isolation level is released as soon as the read statement is completed. It won’t hold onto the end of the transaction. This can cause unrepeatable reads, which means if the reader wants to read the same data twice, between two reads, since the shared lock of the first read has been released, the writer can perform modification on the data, and after the writer releases the exclusive lock, when the reader reads the data again, the results won’t be the same as the results from the first read. In conclusion, READ COMMITTED can prevent dirty reads, but can also lead to unrepeatable reads.

In REPEATABLE READ, the shared lock requested by the reader will hold until the end of the transaction. This means, if in the same transaction two reads happen on the same data resource, between those reads, writer cannot modify the data because the shared lock has not been released yet, and there is an incompatibility between the shared lock and the exclusive lock. Also, in REPEATABLE READ, ‘lost update’ situation will be prevented because it will lead to a deadlock in REPEATABLE READ isolation level.

In SERIALIZABLE, phantom reads can be prevented. In REPEATABLE READ, between reads, new data can be inserted, and if the inserted new data happens to satisfy the reader’s query filter, then when the reader performs a second read with the same query, new rows will be returned, and this is called phantom reads. But in SERIALIZABLE, it will block the attempts made by other transactions to insert new rows that are qualified for the query’s filter.

Another model for isolation is row versioning. In row versioning, we have SNAPSHOT and READ COMMITTED SNAPSHOT. This model is also referred to as optimistic concurrency control. In row versioning, readers don’t need to request for a shared lock to read which means the reader will guarantee to have the results without the wait. If the current version of data is not the one that readers are supposed to see, then the server will return to them the last version of committed rows. For SNAPSHOT, it is similar to SERIALIZABLE level, which means it won’t have dirty reads, unrepeatable reads, and phantom reads. When the transaction begins, if there are UPDATE and DELETE statements, then the server needs to copy data to tempdb for later retrieval. And during the entire transaction, it will retrieve the same data in tempdb to the read to ensure repeatable reads.

For READ COMMITTED SNAPSHOT, it is similar to READ COMMITTED, which means it may have unrepeatable reads and phantom reads, but not dirty reads. Within the same transaction, between reads, the server copies data to tempdb for readers to retrieve in the next read. And since there is no interference between readers and writers, if writers modify data, when the reader performs the next read, the reader might get unrepeatable reads or phantom reads.

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

Tuning Advisor:

The Tuning Advisor in SQL server can optimize a specific SQL query. The Tuning Advisor can provide a lot of great analysis and recommendations based on our queries and workload. It can help us improve the performance of executing SQL queries by recommending structural changes such as indexes, partitioning, etc. I use a specific query example in Question 2 to present the tuning process.

First, choose the query in Question 2 and right click it.

Text

Description automatically generated with medium confidence

In the drop-down list, choose ‘Analyze query in Database Engine Tuning Advisor’. Then, we jump to ‘Database Engine Tuning Advisor’ page and there is a new session created.

Graphical user interface, text, application, email

Description automatically generated

Make sure that we choose the correct database under which the SQL query was executed.

Graphical user interface, text, application, email

Description automatically generated

Then, we click ‘Start Analysis’. It might take a while sometimes.

After the analysis, the Tuning Advisor will render some recommendations. As we can see here, the estimated improvement for this query is 74%.

Graphical user interface, text, application, email

Description automatically generated

It was recommended that there are some objects that needs to be improved upon. The actual definition lies within ‘Definition’ Column. We can also generate some reports of this analysis such as the statement cost report under ‘Reports’ section.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

According to the recommendations, I manually chose the first two in the definition, and added those two definitions back into the query and executed the query again.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

After running the updated query again, we improved the performance a little because the improvement percentage got smaller.

We can also use SQL Server to tune the database to see if there are any recommendations with regards to the changes in physical structure.

Extended events:

We can create extended event sessions to monitor the database. If there is any troubleshooting performance in the database such as executing expensive queries, the extended events can capture it and further analyze it. The specific process is as follows:

Under Management folder, find Extended Events. Inside the Sessions folder under the Extended Events, we can see all the sessions created. The system has three extended events by default: AlwaysOn\_health, system\_health, and telemetry\_xevents. Here, we create a new session.

Graphical user interface, application

Description automatically generated

We name the new session with newSession and configure the session with our needs.

For example, with regards to the events that we want to capture, I chose the ‘alter\_table\_update\_data’, which means this event is captured when an ALTER TABLE ADD column operation results in an UPDATE that affects every row. This event potentially can capture expensive DDL statements. Graphical user interface, application

Description automatically generated

I also set some global fields that I want the session to report if there is any event being captured. After some configuration, here is the summary of my customized configuration of this event sessions.

Graphical user interface, text, application

Description automatically generated

After creating an extend event session, we can start the session, and we can also choose to watch the live data if we want. We run the query in Question 2 again and open the ‘Watch live Data’ option, we can see the session is capturing the transactions as well as the SQL batch.

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, table

Description automatically generated

We can also choose to export the session or generate customized report of the current session. If we want to stop the session, just click ‘Stop Session’.

DMV:

DMV are dynamic management view. It can be found under ‘System Databases – master – Views-System Views’.

Graphical user interface, text

Description automatically generated

The DMVs are used for or understanding and managing SQL Server statistics or health by using the dynamically constructed views. Basically, these views can give us the status of the important things which are helpful in understanding how the server is running.

For example, here I show a DMV named sys.dm\_exec\_procedure\_stats. Graphical user interface, text, application

Description automatically generated

This DMV can give us the information about the stored procedure whenever we execute one. It can give us the information such as when the procedure was run last time, how much it took to execute, etc. To retrieve the information of this DMV, just simply run the query ‘SELECT \* FROM sys.dm\_exec\_procedure\_stats;’. Here is the result shown below.

Graphical user interface, text, application, email

Description automatically generated

Log:

SQL Server maintains its own log files that records transactions and error events. Each committed change in a transaction is recorded in the transaction log. When system fails and restarts, the transaction log can help the server to see if there is any committed change recorded in the transaction log that hasn’t been actually applied to the data before server restarts. If there is, then redo these changes. If there are changes before restart which are not recorded in the log, then rollback these changes. Also, the error Log is a great place to find information about what’s happening on the database server. We can find SQL Server Logs under ‘Management-SQL Server Logs’ as shown below.

Graphical user interface, text, application

Description automatically generated

We can see there is a file serves as the current log file. Every new message is written to that file. The other files are archived files, and they contain previous messages. Each time SQL Server is restarted, the files are being recycled. In other words, it means that a new log file is created and becomes the new current log file. And in the meantime, the oldest archived log file is deleted. And all other log files are pushed back one step, which means the previous current log file now becomes the ‘Archive #1’ file and so on.If we want to see the content of the log files, just click ‘View SQL Log File’. And here it contains all the information about what happened on my SQL server recently as well as capture some error events.

Graphical user interface, text, application

Description automatically generated

Execution Plan:

The execution plan in SQL Server Management Studio is a graphical representation of the steps that are involved in the execution of queries. Once a query is executed, the query processing engine generates multiple execution plans and selects the one that has the best performance. The execution plans include Estimated Execution Plan and Actual Execution Plan. The estimated execution plan is a guess by the query processor about the specific steps needed to return the results. It is often generated before the query has been executed. Whereas the Actual Execution is generated after the query has been executed. It shows the actual operations and steps involved while executing the query. The Actual Execution Plan may or may not differ from the Estimated Execution Plan.

We can find the Estimated Execution Plan and The Actual Execution Plan as shown below.

Before we actually run the query, we can show its Estimated Execution Plan.

Graphical user interface, application, table, Excel

Description automatically generated

After running the query, we can display its Actual Execution Plan.

Graphical user interface, text, application

Description automatically generated

Sometimes, when the query is relatively simple, their estimated query plan and the actual query plan may be the same as shown here. According to the execution plan, we can know that the cost percentage of Query 1 (87%) is much higher than that of Query 2 (13%) which means the first query is more expensive.