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

Generally, a transaction represents any change within a DBMS against a database. The changes can consist of a single read, write, delete, or update operations or a combination of those. ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee the database transactions are processed reliably. Atomicity means all operations in a transaction succeed or every operation is rolled back. Consistency stands for the database is structurally sound on the completion of a transaction. Isolation implies that transactions do not contend with one another. Durability exemplifies the results of applying a transaction are permanent, even in the presence of failures.

SQL Server can operate three different transactions modes: Autocommit Transaction mode, Implicit transaction mode, and Explicit transaction mode. Autocommit transaction is a default mode, and each T-SQL statement is evaluated as a transaction, and they are committed or rolled back according to their results in this mode. If we run a query without mentioning the BEGIN TRAN, it will be considered an implicit transition. On the other hand, if we run a query that starts with BEGIN TRAN and ends with COMMIT or ROLLBACK, it will be considered an explicit transaction. We can also name a transaction and save it use the SAVE TRANSACTION syntax. It can be a save point, and we can use it to roll back any particular part of the transaction rather than the entire transaction.

Concurrency is a situation that arises in a database due to the transaction process. DBMS concurrency is a problem because accessing data simultaneously by two different users can lead to inconsistent results or invalid behavior. SQL Server provides five different levels of transaction isolation to overcome these Concurrency problems. These five isolation levels work on two major concurrency models: the pessimistic model and the optimistic model. In the pessimistic model of managing concurrent data access, the readers can block writers, and the writers can block readers. On the other hand, in the optimistic model of controlling concurrent data access, the readers cannot block writers, and the writers cannot block readers, but the writer can block another writer.

Pessimistic concurrency control utilizes a system of locks to prevent users from modifying data in a way that affects other users. When the database engine processes a statement, the query processor decides which resources need to be accessed and how they will be used. Based on this information, it then determines what types of locks are required to protect each resource. The acquired locks also depend upon the transaction isolation level setting. In short, locks are there to protect resources. Lock modes include Shared, Update, Exclusive, etc.

Shared locksallow concurrent transactions to read a resource, but no other transaction can modify the resource while the lock is held. When transactions might need to update the resource, we use Update locks. Only one transaction can gain an update lock at a time. If data is to be modified, then that lock is converted to an exclusive lock. To ensure that only one transaction can update data at one time, we use Exclusive locks. And nothing else can access that data unless the read uncommitted isolation level is set.

When we connect to a SQL server database, the application can submit queries with one of five different isolation levels. Read Uncommitted, Read Committed, Repeatable Read, and Serializable come under the pessimistic concurrency model. Snapshot comes under the optimistic concurrency model.

Read Uncommitted is the first level of isolation. One transaction is allowed to read the data that is about to be changed by the commit of another process. Read Committed is the second level of isolation and it is the default level. In this level, if you are reading data then the concurrent transactions that can delete or write data, some work is blocked until the other work is complete. Repeatable Read is the third level of isolation. At this level, the transaction has to wait till another transaction's update or read query is complete. But if there is an insert transaction, it does not wait for anyone. Serializable is the highest level of isolation. In this level of isolation, we can ask any transaction to wait until the current transaction completes.

Snapshot takes a snapshot of the current data and uses it as a copy for the different transactions. Each transaction has its copy of data here. If a user tries to perform a transaction like an update or insert, Snapshot will ask him to re-verify all the operations before the process gets started executing.

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 contains a set of processes to optimize the queries to make them run as efficiently as possible. Before doing performance tuning, we have to do performance monitoring first. There are some helpful tools we can use to know how our query performance is. Database Engine Tuning Advisor analyzes databases and gives some recommendations. It helps with a series of things include:

1. Troubleshooting the performance of a specific problem query

2. Tuning a large set of queries across one or more databases

3. Performing exploratory what-if analysis of potential physical design changes

4. Managing storage space.

An example of using Database Engine Tuning Advisor is shown in Fig 1.

Graphical user interface, text, application

Description automatically generated

Figure Results of Database Engine Tuning Advisor

Extended Events provide a set of methods for collecting different events from SQL Server and correlating those different events within a single tool. It’s possible to grab:

1. Deadlocks + waits
2. Waits + lock graph
3. TempDB spill + query plan

Dynamic Management Views (DMVs) are another great tools to help troubleshoot performance-related issues. Microsoft introduced them in SQL 2005 and added additional DMVs to help troubleshoot issues with each new release. The DMVs act like any other view where you can select data from them, and the DMFs (dynamic management functions) require values to be passed just like any other function.

We can keep logs and data files separated for performance tuning. Log and data files need to be stored on separate drives as the writing and accessing of log files is sequential, whereas writing and accessing of data files is non-sequential. Storing log and data files separately onto other physical drives can enhance the system’s performance levels.

SQL Server Execution Plan is a binary representation of the steps that will be followed by the SQL Server Engine to execute the query. We can use it to know is if an index will help and to do performance tuning. Fig 2-4 show the difference before and after creating nonclustered index.

Text

Description automatically generated

Figure Query to do performance tuning

A picture containing graphical user interface

Description automatically generated

Figure Execution Plan before creating nonclustered index

Diagram

Description automatically generated

Figure Execution Plan after creating nonclustered index

30. Write a short essay talking about a scenario: Good news everyone! We (Wide World Importers) just brought out a small company called “Adventure works”! Now that bike shop is our sub-company. The first thing of all works pending would be to merge the user logon information, person information (including emails, phone numbers) and products (of course, add category, colors) to WWI database. Include screenshot, mapping and query.

First, we loaded the AdventureWorks sample databases from this website: https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms. Second, we inserted data into "Application. People" in WWI from "Person. Person" in AdventureWorks joined with multiple related tables shown in Fig 5. Third, we inserted data into "Warehouse. Colors", "Purchasing. Suppliers", and “Warehouse.StockGroups” in WWI from "Production. Product" and "Production.ProductCategory" in AdventureWorks shown in Fig 6. Forth, we created a temporal table to save the query results from joined multiple tables related to product information in AdventureWorks and inserted them into "Warehouse.StockItems" in WWI shown in Fig 7. Last, we insert the information to the joined table, “Warehouse.StockItemStockGroups”, in WWI from "Warehouse.StockItems" and “Warehouse.StockGroups” as Fig 8. We met some problems include the different server collations in the two databases, and the duplicated product names that we added the ROW\_NUMBER() function to solve.

Text

Description automatically generated

Figure Insert data into People

A screenshot of a computer

Description automatically generated with medium confidence Graphical user interface, text, application

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

Figure Insert data into Colors, Suppliers, and StockGroups

Text

Description automatically generated

Figure Insert data into StockItems

Text

Description automatically generated

Figure Insert data into StockItemStockGroups