

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

Transaction is a unit of work, including all kinds of query and activities to modify the data. By default, each statement is a transaction. And we can add multiple statement into one unit of work by define a transaction.

Transaction has four properties:

Atomicity: which means statements in a transaction can only be committed or rollback together.

Consistency: which means data should be in a consistent state for all user during a transaction start and end.

Isolation: Transactions can not see what others are doing, they access only intermediate data.

Durability: Changes made by transaction are not able to undo.

Locks are applied on resource to prevent others from accessing it. It avoid the conflicts and ensure the consistency of transaction. Locks interact between transactions help us implement the isolation level.

Isolation level the consistency level of data you want the transaction to access.

There are four different isolation level:

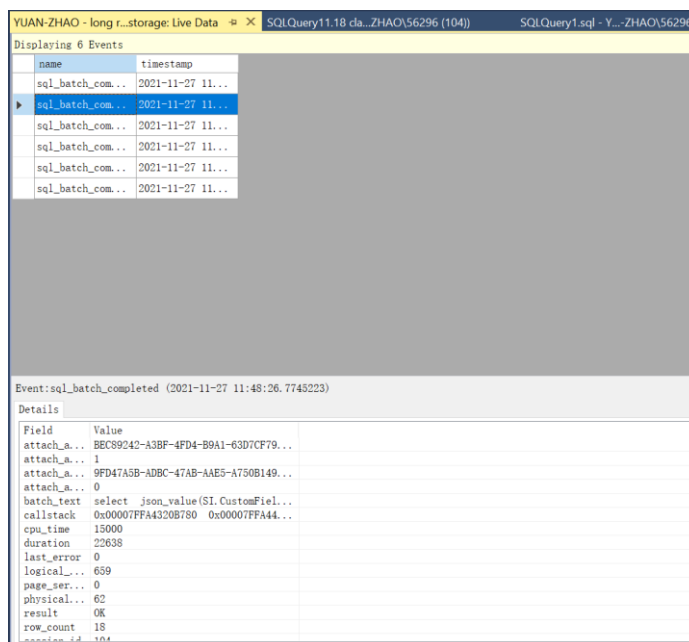
Read Uncommitted is the lowest level that has no lock on it. Transaction can dirty read the uncommitted data. Read committed is the default isolation level of SQL server that prevent dirty read. In this level, reader got the shared lock after writer committed the transaction.

The REPEATABLE READ is a higher isolation level that ensure the consistency between reads in same transaction. Under this condition, locks applied to prevent other locks before reader end the transaction.

The SERIALIZABLE is the highest level that prevent phantom reads of data that changed between the reads of transaction. This means lock are applied not only on the current rows if resource but also the future rows in the range of filter keys.

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.

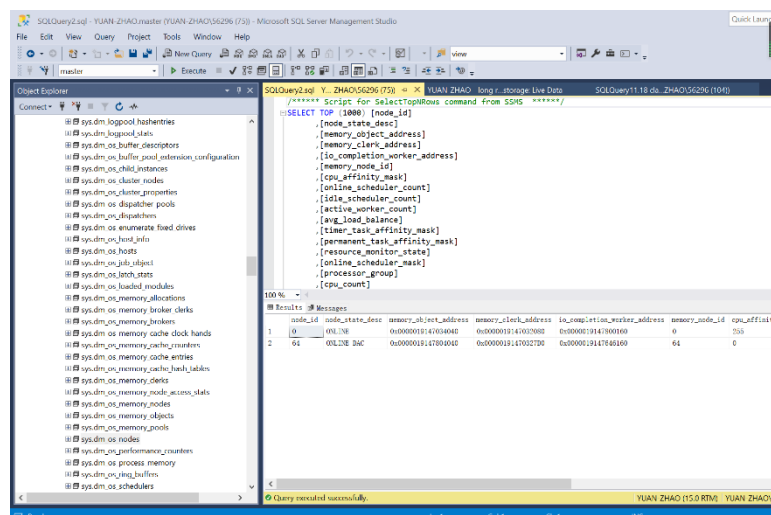
Logs record the status of system. There are several tools that let the user to access the system log. Extended events use the event bubbling system to capture all the event from bottom to top layer. IT helps you know which query is giving you the bottleneck. SQL server logs is another way to do this. Using DBCC queries to start tracing on certain flags of events that will appear in the system log.



The screenshot shows the SQL Server Event Viewer. The top pane displays a list of 6 events, all named 'sql_batch_completed' with timestamps from 2021-11-27 11:48:26.7745223. The bottom pane shows the details for the selected event.

Field	Value
attach_a...	BEC89242-A3BF-4FD4-B9A1-63D7CF79...
attach_a...	1
attach_a...	9FD47A5B-ADBC-47AB-AAE5-A750B149...
attach_a...	0
batch_text	select json_value(SI.CustomFiel...
callstack	0x00007FFA4320B780 0x00007FFA44...
cpu_time	15000
duration	22638
last_error	0
logical...	659
page_ser...	0
physical...	62
result	OK
row_count	18
...	...

DMV is the system dynamic management view is a collection of multiple views in master database. It can tell you all the status of the system. And we can use Performance monitor to check how the system resource is being use and the performance.



The screenshot shows SQL Server Enterprise Manager with a query executed against the 'master' database. The query selects the top 1000 nodes from the 'sys.dm_os_nodes' DMV. The results pane shows two rows of data.

node_id	node_state_desc	memory_object_address	memory_clerk_address	io_completion_worker_address	memory_node_id	cpu_affinity
1	ONLINE	0x0000015140340402	0x0000015140340402	0x0000015140340402	0	255
2	ONLINE	0x0000015140340402	0x0000015140340402	0x0000015140340402	64	0

In the execution plan, we can find which steps of execution takes larger percentage of the execution time, or which part we can make improvement for example reduce the amount of table scan. When we find some missing index that led to table scans, Tuning advisor is one of the best ways that let the system automatically add those missing index.

```

PIVOT(
COUNT(t.aat) for StockGroupName in ([Air-line Novelties],[Clothing],[Computing Novelties],[Furry Footwear],[Hugs],[Novelty Items]
)
) AS pivot_table;

-- List of customers and their ordered stockitems totals (unit price * quantity) on 2014-12-31 (using the unit price of that day)
select sc.CustomerName,ws.StockItemName,sum(sol.Quantity)*ws.UnitPrice as total
from sales.Customers FOR SYSTEM_TIME AS OF '2014-12-31' sc
join Sales.Orders so
on sc.CustomerID=so.CustomerID and so.OrderDate = cast('2014-12-31' as date)
join Sales.OrderLines sol
on sol.OrderID=so.OrderID
join Warehouse.StockItems FOR SYSTEM_TIME AS OF '2014-12-31' ws
on sol.StockItemID=ws.StockItemID
group by sc.CustomerName,ws.StockItemName,ws.UnitPrice;

```

100% - 4 Messages 8 Execution plan

Query 1: Query cost (relative to the batch): 100%

select sc.CustomerName,ws.StockItemName,sum(sol.Quantity)*ws.UnitPrice as total from sales.Customers FOR SYSTEM_TIME AS OF '2014-12-31' sc join Sales.Orders so on sc.CustomerID=so.CustomerID and so.OrderDate = cast('2014-12-31' as date) join Sales.OrderLines sol on sol.OrderID=so.OrderID join Warehouse.StockItems FOR SYSTEM_TIME AS OF '2014-12-31' ws on sol.StockItemID=ws.StockItemID group by sc.CustomerName,ws.StockItemName,ws.UnitPrice;

Missing Index (Impact 60.32%): CREATE NONCLUSTERED INDEX (<Name of Missing Index, sysname,>) ON [Sales].[Orders] ([Ord

Operator	Cost	IO	IO Stalls	IO Stalls (s)	IO Stalls (ms)	IO Stalls (us)	IO Stalls (ns)
Compute Scalar	Cost: 0 %	0.01s	273 rd	728 (37%)	743 (36%)	743 (36%)	743 (36%)
Stream Aggregate (Aggregate)	Cost: 1 %	0.00s	273 rd	743 (36%)	743 (36%)	743 (36%)	743 (36%)
Sort	Cost: 0 %	0.00s	273 rd	743 (36%)	743 (36%)	743 (36%)	743 (36%)
Hash Match (Inner Join)	Cost: 0 %	0.00s	273 rd	743 (36%)	743 (36%)	743 (36%)	743 (36%)
Constellation	Cost: 0 %	0.00s	273 rd	743 (36%)	743 (36%)	743 (36%)	743 (36%)
Clustered Index Scan (C- [StockItems].[WH_Wareho...])	Cost: 0 %	0.00s	273 rd	743 (36%)	743 (36%)	743 (36%)	743 (36%)
Clustered Index Seek (C- [StockItems].[Archive].[I...])	Cost: 0 %	0.00s	273 rd	743 (36%)	743 (36%)	743 (36%)	743 (36%)