In-Memory OLTP Demo Script

**Demo Overview: Talking Points**

The key takeaways are:

1. In-Memory OLTP is fast
   1. High end system performance (1 million rows / second) on mid-range system
   2. Up to 30X or more comparing to the traditional SQL implementation
2. In-Memory OLTP is built into SQL Server natively
   1. Flexibility for staged conversion by first convert the data and then the business processing

**Demo steps**

|  |  |  |
| --- | --- | --- |
| Screenshot | Talking Points | Demo action |
|  | *Introduce a business scenario where airline ticket reservation is running slow under high load (think customers rebooking due to severe weather) .We have a* ***simple*** *simulator to simulate the load (80 client threads simulating a scaled-out mid-tier)* | * *Two screens*   + *Driver*   + *SSMS – for running the analysis report (right click on MDW->reports->Transaction Performance Analysis Overview* |
|  | *Run the load in SQLServer as-is and use this as a* ***baseline****. The OLTP throughput is limited at ~2400 transactions per second, which is not meeting business requirements for peak times, e.g., severe weather situations* | * *Click on Start button to run the load on a traditional SQL engine* * *The driver shows the TPS in a speedometer and histogram form to capture the history.*   *TPS: Transactions per second, this is the customer workload throughput, the higher the better.* |
|  | *Look at the diagnostics and see CPU at ~40% and system indicates heavy internal contention which is difficult to resolve.* | * *CPU% - system utilization, the higher the better* * *Latch waits – system contention, the lower the better* |
|  | *We have the Analysis tool to find out what In-Memory OLTP can help with* | * *Open SSMS, the report should have been preloaded*   + *If not pre-loaded, go to databases->MDW->Reports -> Transaction Performance Analysis Overview* |
|  | *We ran an analysis tool on the production system, and it indicates the table which has the most* ***usage*** *and contention, as well as the ease of migration. Top right quadrant is the sweet spot for memory optimized tables. This shows “TicketReservationDetail” is the prime candidate for migration.* | * Click on “Tables Analysis” -> “Usage Analysis” -> first timestamp link   *Optional: can skip contention analysis – which shows contention on the same table “TicketReservationDetail”* |
|  | *In this step, we run a script to convert this table into a memory optimized table.* | *UI Migrate -> Step 1 Migrate Table* |
|  | *Run the load again, and we observe a* ***~6-7X throughput gain*** *with CPU at ~100% - and notice the contention is gone. The gain is due to CPU usage efficiencies, as well as optimized data access in memory. Note that this has the full ACID guarantees and no queries need to be changed, and this did not require are application changes.* | * *Click Run* * *Let it run for 10-15 seconds and click stop* |
|  | *Wait – we can do* ***better****, In-Memory OLTP also utilizes each CPU cycles more efficiently by natively compiling the stored procedures, and the analysis tool identified those high cost SPs running the business logic such as “InsertReservationDetails”* | * Click on back button to go back to AMR entry portal * Click on “Stored Procedure Analysis” -> “Usage Analysis” -> first timestamp link |
|  | *In this step, we run another script to convert top 2 high cost SPs into In-Memory OLTP to take advantage of native compilation* | *UI Migrate -> Step 2 Migrate SP* |
|  | *And we observed yet additional ~4X gain! Now the system is running at about* ***25-28X of the baseline*** *on the same HW.* | * *Click Run* * *Click Stop before 30 seconds* |
|  | *Summary of the steps and gains.*  *Remind people that this is a* ***mid-range server producing a high end system performance*** *(2 sockets 12 cores, and it’s producing more than 2M row writes and 2M row reads per second).* | *No need for the slide any more, since the history is captured cleanly in the graph.* |
|  | *Optional steps*  *Use beyond compare to show the code change* | *Control-click to select left and right files to compare*  *To show inter-op changes: Step1\_diskbased.sql <> Step2\_interop.sql*  *To show native compilation changes (incrementally from interop): Step2\_interop.sql <> Step3\_native.sql* |