Principles of performance tuning

• The Goal:
  • Minimum response time and Maximum throughput
  • Reduce network traffic, disk I/O, memory usage and CPU time
  • This is achieved through understanding:
    • Application requirements
    • Logical and physical structure of the data
    • Tradeoffs between conflicting uses of the database. (OLTP vs. DSS)

• Start optimizing in early stage of development. It tends to be much harder later.
Principles of performance tuning

- Good design will reduce performance problems:
  - Architecture
  - Application & queries
  - Database
  - Hardware
- Identify the areas that will yield the largest performance boosts
  - Over widest variety of situations
  - Focus attention on these areas
- Consider Peak load, Not average ones.
  - Performance issues arise when loads are high

Measuring Performance and Detecting Bottlenecks
Measuring Performance

• The procedure
  1. Create a baseline
  2. Find bottlenecks
  3. Tune system
  4. Compare performance with baseline
  5. Repeat

Measuring Performance

• Performance Monitor
• SQL Traces & SQL Server Profiler
  • And derivatives
• Extended Events
• SQL Server Management Studio
• Activity Monitor
• Dynamic Management Views
• Database Tuning Advisor
Performance Tuning Tools

Performance Monitor
Performance Monitor

- Primarily used to discover hardware bottlenecks
- Can display values that relate to processor, memory and disk activities
- And many more counters...

Tips
- Run during heavy load
- Run it periodically
- Compare with historical data to analyze server activity trends.

Demo: Perfmon Basics
• Processor: % Processor Time
  • If more than 80-90% for long periods, you might have a CPU bottleneck.

• System: Processor Queue Length
  • Should be less than 4 per processor.
  • If higher for long periods, you have CPU problem.

• Network Interface: Bytes received/sec
• Network Interface: Bytes sent/sec
  • Compare to adapter bandwidth

• Memory: Memory: Page Faults/sec, pages/sec
  • Monitor disk paging which might cause high disk usage and thus, reduce performance significantly.

• SQL Server: Memory Manager: Total Server Memory
• ,Target Server Memory
  • Counters should be identical or close

• SQL Server: Buffer Manager: Buffer Cache Hit Ratio
  • Should be 90% or higher
### Hardware Counters in Performance Monitor

- **PhysicalDisk: Avg. Disk sec/read, sec/write**
  - Determines how long a read or write operation takes.
  - Should be lower than 20ms (0.02s) at all times
- **PhysicalDisk: Avg. Disk reads/sec, writes/sec**
  - Determines how many read or write operations occur.
  - A single 15krpm disk supports ~180 iops
- **PhysicalDisk: Current Disk Queue Length:**
  - Watch out for high values.
- **SQL Server: Buffer Manager: Page Reads/sec, Page Writes/sec**
  - Monitor disk activity caused by SQL server

### SQL Server Counters in Performance Monitor

- **SQLServer: Access Methods: Full scans/sec**
  - How many tables scans occur per sec
- **SQLServer: SQL Statistics: Batch Requests/sec**
  - How many batches are executed per sec
  - Compare to next counter (should be higher)
- **SQLServer: SQL Statistics: Compilations/sec**
  - How many plans are compiled per sec
  - Compare to next counter (should be higher)
- **SQLServer: SQL Statistics: Recompilations/sec**
  - How many plans are recompiled per sec
SQL Server Counters in Performance Monitor

• SQLServer: Locks : Lock requests/sec
• SQLServer: Locks : Lock waits/sec
• SQLServer: Locks : Average Wait Time (ms)
  • Lock statistics (requests vs. waits)

• And many more
  • Mirroring
  • Replication
  • Service broker
  • Fulltext Search
  • SQL Agent
  • …

Demo: Perfmon for Baselines
Performance Counters Analysis

- Perfmon
- PAL/Relog
  - Console applications for perfmon logs analysis
  - Use to load performance log into a SQL Server database
  - Continue analysis with queries & reports (Excel/SSRS)
  - Use aggregate functions for minimum, maximum and average values.
  - Investigate work-hours vs. off-hours, weekdays vs. weekends

Demo: PAL
Demo: Resource Monitor

Demo: Custom SQL Counters
SQL Trace

• Instruments the server to generate events
  • Event producer is integrated in SQL server:
    • Connection Manager
    • Relational Engine
    • Storage Engine
    • Lock and Log Manager
  • Provides a general purpose trace event model
    • Producers (SQL Server, Analysis Services)
    • Consumers (SQL Profiler, SMO Based applications)

SQL Trace Event Examples

• Capture detailed data from:
  • Connection Manager
    • Batch, RPC, Connection, Disconnect
  • Relational Engine
    • Stored Procedures, SQL statements, Query plans
  • Lock manager
    • Locks, deadlock graph,
  • Auditing
    • Object access (success/failure)
    • Object creation
    • Permissions grant/deny/revoke
SQL Trace

- Powerful (server side) filtering
  - Thresholds, ranges and equalities
    - Duration > 1000
    - SPID in (10,11,12) (expressed as multiple ==)
  - Include / exclude
    - SQL like ‘exec sp_trace%’
    - Application name not like ‘%profiler%’
  - You may filter by any column
    - DBID, Reads / Writes, Roll name, Severity...

SQL Trace

- Totally configurable
  - Which events to trace
  - Which data to collect (columns)
  - Pick your destination
    - Trace File (.TRC)
      - Server side
      - Client side (via consumer)
    - Consumer application (e.g. SQL Profiler)
    - SQL Server table (via SQL Profiler)
SQL Trace

- Other features:
  - File roll-over on size limit
  - Stop traces based on time and/or size
  - Auto-start traces (with jobs or stored procedures)
SQL Server Profiler

- Used to start, stop, view and analyze SQL traces
- Adds overhead to the low-overhead trace
- Helps discover:
  - Expensive queries
  - Very large sort & hash operations
  - Table searches & updates
  - Index usage
  - Missing join predicates
  - Missing statistics

SQL Server Profiler

- Trace management / analysis client
- Input sources:
  - Real time trace
  - Trace file/table
- Analytical capabilities
  - Filtering, aggregation and sorting
    - find the worst queries
    - counts of events by class
  - Grouping
    - By Application name / host name / SPID

Microsoft
SQL Server Profiler

- Debugging & QA
  - Trace SQL exceptions

  - Multi-user replay
    - Use the template “TSQL_Replay”
    - Display resultsets, errors
    - Measure playback duration
    - Allows breakpoints & debugging

SQL Trace Tips

- Create your own templates and share them
  - Tdf files in \Tools\Profiler\Templates\Microsoft SQL Server
- Be selective – don’t trace everything
- The events you would normally trace are RPC:Completed and SQL:Batch Completed
- The columns you would usually trace are: TextData, CPU, duration, reads, writes, StartTime, EndTime,
- You should also consider adding DatabaseName, HostName, RowCounts
- Play with your filters until you get the right balance – enough data to analyze, but not too much data.
SQL Trace Tips

• Use Profiler to define your trace.

• Don’t use Profiler to start the trace.

• Instead, save trace definition to script and execute the script to get a low-overhead server side script.

• Compare SQL traces before & after performance tuning, to show improvement.

Demo: Trace Best Practices
SQL Trace Analysis

- Use SQL Profiler for line-by-line analysis

- To find offending queries & stored procedures:
  - Load into a table and aggregate by Textdata
  - Use free tools such as ClearTrace or DBSophic Trace Analyzer

- Get a top 10 list of offending queries, handle them one by one, collect another trace and repeat the process until requirements are satisfied

Demo: Tracing Blocks
Dynamic Management Objects

- Dynamic management views and functions (DMV) return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance.
- In General, statistics returned by DMVs is not persisted.
- Low overhead
  - Many DMVs expose information that needs to be maintained anyway.
## DMV Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Language Runtime (CLR)</td>
<td><code>Sys.dm_clr_*</code></td>
</tr>
<tr>
<td>Database</td>
<td><code>Sys.dm_db_*</code></td>
</tr>
<tr>
<td>Indexing</td>
<td><code>Sys.dm_db_index_*</code></td>
</tr>
<tr>
<td>Database Mirroring</td>
<td><code>Sys.dm_db_mirroring_*</code></td>
</tr>
<tr>
<td>Execution</td>
<td><code>Sys.dm_exec_*</code></td>
</tr>
<tr>
<td>Full-Text Search</td>
<td><code>Sys.dm_fts_*</code></td>
</tr>
<tr>
<td>I/O</td>
<td><code>Sys.dm_io_*</code></td>
</tr>
<tr>
<td>Query Notifications</td>
<td><code>Sys.dm_qn_*</code></td>
</tr>
<tr>
<td>Replication</td>
<td><code>Sys.dm_repl_*</code></td>
</tr>
<tr>
<td>Service Broker</td>
<td><code>Sys.dm_broker_*</code></td>
</tr>
<tr>
<td>SQL Server Operating System</td>
<td><code>Sys.dm_os_*</code></td>
</tr>
<tr>
<td>Transactions</td>
<td><code>Sys.dm_tran_*</code></td>
</tr>
</tbody>
</table>

## DMV Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Data Capture</td>
<td><code>Sys.dm_cdc_*</code></td>
</tr>
<tr>
<td>Object</td>
<td><code>Sys.dm_sql_*</code></td>
</tr>
<tr>
<td>Resource Governor</td>
<td><code>Sys.dm_resource_governor_*</code></td>
</tr>
<tr>
<td>SQL Server Extended Events</td>
<td><code>Sys.dm_xe_*</code></td>
</tr>
<tr>
<td>Security</td>
<td><code>Sys.dm_cryptographic_*</code></td>
</tr>
<tr>
<td></td>
<td><code>Sys.dm_provider_*</code></td>
</tr>
<tr>
<td></td>
<td><code>Sys.dm_audit_*</code></td>
</tr>
</tbody>
</table>
DMV Examples

- Execution plans
  - sys.dm_exec_query_stats
  - sys.dm_exec_sql_text
  - sys.dm_exec_cached_plans
- Indexes
  - sys.dm_db_index_operational_stats
  - sys.dm_db_index_physical_stats
  - sys.dm_db_missing_index_details
- Database I/O
  - sys.dm_db_file_space_usage
  - sys.dm_io_virtual_file_stats

Demo: DMVs and sys2 Enhancements
SSMS

- Activity Monitor
  - New features!
- Built in reports
- Execution plans
- Time and IO statistics

Demo: SSMS Tips & Tricks
Database Tuning Advisor

An automatic tuning tool that copes with:

- Retrieval, update & cursor operations
- Multiple databases (single server)
- Optimizer’s plan choices & cost functions
- Single- and multi-column indexes
- Indexes for foreign keys
- Disk space increase & decrease
- “Only add” vs. “replace & add” indexes
- Partitioning & Indexed views  
  (Enterprise edition only)
Database Tuning Advisor

- Goal: Determine Optimal Index Requirements based on query statements
- Steps:
  - Analyze your SQL workload
    - SQL Script file
    - Trace file
  - Tune a workload
    - Runs a sample selection of queries from the workload through the Query Processor with different index combinations and compares query costs
  - Recommend changes
    - Adding, dropping or changing indexes
    - Partitioning
  - Implement changes / save to script

DBA has full control over tuning effort, disk space, etc.
DTA Tips

- Use with caution!
- Treat recommendations as such, not all should be implemented.
- Creating many indexes isn’t always better.
- It’s always easier to add indexes than to remove DTA recommended indexes later.
Introducing SQL Server 2012 Extended Events Enhancements

Keren Bartal
Tzahi Hakikat
888 holdings

Agenda

• About us
• Introduction to Extended Events
• Extended Events 2008
• Extended Events Practical Terminology
• Extended Events 2012 Enhancements
• Summary
About 888

- 888.com is a global online gaming company.
- Our purpose is to provide quality entertainment for people who enjoy gambling.
- Giving them the opportunity to do so in a safe, fun, fair, regulated and secure environment.
### 888 Database Environment

- **50 Production Instances**
- **300 Development Instances**
- **400 Databases**
- **250 TB Of Data**
- **24*7 Availability**
- **99.95 Uptime**
Agenda

• About us
• **Introduction to Extended Events**
• Extended Events 2008
• Extended Events Practical Terminology
• Extended Events 2012 Enhancements
• Summary

Extended Events

• General event-handling system for windows servers
• Used for problem diagnosis and info gathering and auditing
• The Extended Events infrastructure supports the correlation of data from SQL Server and OS
Extended Events

- Support 7 different types of targets
- Event and consumer agnostic
  - Any event can be processed by any consumer
  - New events can be added, immediately useable
- Rich predicate system for filtering
- Less overhead than server-side trace queues
  - 10,000 events processed will consume 1% of single 2GHz processor

Agenda

- About us
- Introduction to Extended Events
- Extended Events 2008
- Extended Events Practical Terminology
- Extended Events 2012 Enhancements
- Summary
Extended Events 2008

drawbacks

• XE required extensive understanding of system catalog views and DMVs
• Event Sessions could only be managed through the use of DDL commands
• Reading target data requires the use of XQuery
Extended Event Metadata

- Catalog views for defined session info
  - server_event_sessions
  - server_event_session_target
  - server_event_session_fields
  - server_event_session_actions
  - server_event_session_events

- DMVs for Event System Metadata
  - dm_xe_package
  - dm_xe_objects
  - dm_xe_object_columns
  - dm_xe_map_values

- DMVs for currently active session info
  - dm_xe_sessions
  - dm_xe_session_targets
  - dm_xe_events
  - dm_xe_event_actions
  - dm_xe_object_columns

Demo

Capture errors with XE 2008

- Find events and actions
- Create a new event session
- View the output
**Agenda**

- About us
- Introduction to Extended Events
- Extended Events 2008
- **Extended Events Practical Terminology**
- Extended Events 2012 Enhancements
- Summary

**Extended Events Objects**

- Module
- Packages
- Events
- Targets
- Actions
- Types
- Predicates
- Maps
Packages

- Packages are metadata containers
- Packages register at module load time
- 9 available packages
  - package0 - XE system objects (default)
  - sqlserver - SQL Server related objects
  - sqlos - SQL Server Operating System (SQLOS) related objects
- SQL audit uses private XE package

Events

- An event is a well known point in code
- Unique schema for each event
- Supports optional fields
- Events fire synchronously
- 264 events in 2008 R2
- 618 events in 2012
**Actions**

- Programmatic response or series of responses to an event
- Can be added to any event
- Adds data to the event payload
- Actions are invoked synchronously
- Trigger a memory dump

---

**Demo**

**Capture errors using the XE UI**

- Create an event session
- Configure action
- Watch live data
Targets

- Target is an event consumer
  - Can be synchronous or asynchronous
- Target types
  - event_file
  - event_counter
  - histogram
  - etw_classic_sync_target
  - pair_matching
  - ring_buffer
  - event_stream

Demo

Monitor locks
Present different types of targets

- Ring buffer
- Event file
- Event counter
- Histogram
- Pair Matching
- Etw_classic_sync_target
Predicates

- Predicates are a set of logical rules that are used to evaluate events when they are processed.
- Boolean expressions using flexible operators
  - Event data
  - Action data
  - Global State
Event Session

- The materialization of combination of metadata elements of XE architecture
- Multiple targets per session
- Event can be in many sessions
  - Actions/Predicates are per event
- Event Session can specify what to do if target can't keep up
- Event Session defines data retention
- Event session can add or remove events on runtime
Event life cycle

Pre-Collect
- IsEnabled check

Collection
- Customizable attribute check
- Event data collected

Predicate evaluation
- Predicate evaluation

Publish
- Actions executed
- Synchronous targets served
- Event data buffered for asynchronous targets

Agenda

- About us
- Introduction to Extended Events
- Extended Events 2008
- Extended Events Practical Terminology
- **Extended Events 2012 Enhancements**
- Summary
Extended Events 2012 Enhancements

• User Interface
  – Advanced & Wizard UI for creating and managing
  – Display & Analysis

• Expanded to other systems
  – Analysis Services, Replication, PDW

• Managed code
  – Powershell object model for runtime and metadata
  – Reader API for XEL files and near real time stream

User Interface

• Event Session list
  – Provides a list of Event Sessions

• New Session Wizard
  – Provides a simplified experience for creating an Event Session

• Extended Events display
  – Tabbed windows that display Extended Events trace data
Demo

Capture queries and group by query hash

- Grouping
- Aggregation
- Save XE to a table

Extended Events Management API

- Management API provides the ability to create and modify event sessions
- Provides a complete object model for XE usage by managed applications
- Provides a XEReader API for reading event files and event streams coming from a running event session on a server
Agenda

• About us
• Introduction to Extended Events
• Extended Events 2008
• Extended Events Practical Terminology
• Extended Events 2012 Enhancements
• Summary

Extended Event Use Cases

• Proactive monitoring
  — Application errors
  — Errors log
  — Event grouping
• Troubleshooting
  — Page Split
  — blocking
• Audit
  — Monitor the access of privileged and non privileged users
The Profiler’s grave

Summary

- SQL Server 2012 offers simplified diagnostic tracing with Extended Events
  - Management Studio integration provides SQL Server Profiler functionality for Extended Events allowing Event Sessions to be created, modified, and scripted
  - Management API allows managed applications to be developed that leverage Extended Events
New performance features in SQL Server 2012

ColumnStore Indexes
Improved Data Warehouse Query performance

- Columnstore indexes provide an easy way to *significantly* improve data warehouse and decision support query performance against very large data sets
- Performance improvements for “typical” data warehouse queries from 10x to 100x
- Ideal candidates include queries against star schemas that use filtering, aggregations and grouping against very large fact tables

What Happens When...

- You need to execute high performance DW queries against very large data sets?
  - In SQL Server 2008 and SQL Server 2008 R2
    - OLAP (SSAS) MDX solution
    - ROLAP and T-SQL + intermediate summary tables, indexed views and aggregate tables
      - Inherently inflexible
What Happens When...

• You need to execute high performance DW queries against very large data sets?
  • In SQL Server 2012
    • You can create a columnstore index on a very large fact table referencing all columns with supporting data types
      • Utilizing T-SQL and core Database Engine functionality
      • Minimal query refactoring or intervention
    • Upon creating the columnstore index, your table becomes “read only” – but you can still use partitioning to switch in and out data OR drop/rebuild indexes periodically

How Are These Performance Gains Achieved?

• Two complimentary technologies:
  • Storage
    • Data is stored in a compressed columnar data format (stored by column) instead of row store format (stored by row).
      • Columnar storage allows for less data to be accessed when only a sub-set of columns are referenced
      • Data density/selectivity determines how compression friendly a column is – example “State” / “City” / “Gender”
      • Translates to improved buffer pool memory usage
How Are These Performance Gains Achieved?

- Two complimentary technologies:
  - New “batch mode” execution
    - Data can then be processed in batches (1,000 row blocks) versus row-by-row
    - Depending on filtering and other factors, a query may also benefit by “segment elimination” - bypassing million row chunks (segments) of data, further reducing I/O
Batch Mode

- Allows processing of 1,000 row blocks as an alternative to single row-by-row operations
  - Enables additional algorithms that can reduce CPU overhead significantly
  - Batch mode “segment” is a partition broken into million row chunks with associated statistics used for Storage Engine filtering

Batch Mode

- Batch mode can work to further improve query performance of a columnstore index, but this mode isn’t always chosen:
  - Some operations aren’t enabled for batch mode:
    - E.g. outer joins to columnstore index table / joining strings / NOT IN / IN / EXISTS / scalar aggregates
  - Row mode might be used if there is SQL Server memory pressure or parallelism is unavailable
  - Confirm batch vs. row mode by looking at the graphical execution plan
Columnstore format + batch mode Variations

- Performance gains can come from a combination of:
  - Columnstore indexing alone + traditional row mode in QP
  - Columnstore indexing + batch mode in QP
  - Columnstore indexing + hybrid of batch and traditional row mode in QP

Creating a columnstore index

- T-SQL
- SSMS
Good Candidates for Columnstore Indexing

• Table candidates:
  • Very large fact tables (for example – billions of rows)
  • Larger dimension tables (millions of rows) with compression friendly column data
  • If unsure, it is easy to create a columnstore index and test the impact on your query workload

• Query candidates (against table with a columnstore index):
  • Scan versus seek (columnstore indexes don’t support seek operations)
  • Aggregated results far smaller than table size
  • Joins to smaller dimension tables
  • Filtering on fact / dimension tables – star schema pattern
  • Sub-set of columns (being selective in columns versus returning ALL columns)
  • Single-column joins between columnstore indexed table and other tables
Defining the Columnstore Index

- **Index type**
  - Columnstore indexes are always non-clustered and non-unique
  - They cannot be created on views, indexed views, sparse columns
  - They cannot act as primary or foreign key constraints

- **Column selection**
  - Unlike other index types, there are no “key columns”
    - Instead you choose the columns that you anticipate will be used in your queries
    - Up to 1,024 columns – and the ordering in your CREATE INDEX doesn’t matter
  - No concept of “INCLUDE”
  - No 900 byte index key size limit

- **Column ordering**
  - Use of ASC or DESC sorting not allowed – as ordering is defined via columnstore compression algorithms

Supported Data Types

- **Supported data types**
  - Char / nchar / varchar / nvarchar
    - (max) types, legacy LOB types and FILESTREAM are not supported
  - Decimal/numeric
    - Precision greater than 18 digits NOT supported
  - Tinyint, smallint, int, bigint
  - Float/real
  - Bit
  - Money, smallmoney
  - Date and time data types
    - Datetimeoffset with scale > 2 NOT supported
Demo: Simple ColumnStore Index

Once built, the table becomes “read-only” and INSERT/UPDATE/DELETE/MERGE is no longer allowed

ALTER INDEX REBUILD / REORGANIZE not allowed

Other options are still supported:
- Partition switches (IN and OUT)
- Drop columnstore index / make modifications / add columnstore index
- UNION ALL (but be sure to validate performance)
Limitations

- Columnstore indexes cannot be used in conjunction with
  - Change Data Capture and Change Tracking
  - Filestream columns (supported columns from same table are supported)
  - Page, row and vardecimal storage compression
  - Replication
  - Sparse columns
- Data type limitations
  - Binary / varbinary / ntext / text / image / varchar (max) / nvarchar (max) /
    uniqueidentifier / rowversion / sql_variant / decimal or numeric with precision > 18
    digits / CLR types / hierarchyid / xml / datetimeoffset with scale > 2
- You can prevent a query from using the columnstore index using the
  IGNORE_NONCLUSTERED_COLUMNSTORE_INDEX query hint

Demo: ColumnStore and Partition Switch
Summary

- SQL Server 2012 offers significantly faster query performance for data warehouse and decision support scenarios
  - 10x to 100x performance improvement depending on the schema and query
    - I/O reduction and memory savings through columnstore compressed storage
    - CPU reduction with batch versus row processing, further I/O reduction if segmentation elimination occurs
  - Easy to deploy and requires less management than some legacy ROLAP or OLAP methods
    - No need to create intermediate tables, aggregates, pre-processing and cubes
  - Interoperability with partitioning
  - For the best interactive end-user BI experience, consider Analysis Services, PowerPivot and Crescent
Improvements for Table Partitioning

- SQL Server 2012 RTM supports up to 15,000 partitions
  - No need for a service pack to gain functionality
- Partition statistics are created using a row sub-set sampling when an index is rebuilt or created - versus scanning all rows to create the statistics
- Additional partition management wizard options can assist with executing or scripting out common partition operations
- Partitioning can be used in conjunction with tables that have a columnstore index in order to switch in and out data

---

On Mon, Oct 1, 2012 at 11:10 PM, Matt Duda <matt@yoursite.com> wrote:

Hi all,

I don’t see much written about the new 15K partition support in SQL Server 2012. I think this must be one of the most useful features which is going under the radar, isn’t it?

My _db_partitioned_ partitions most mentions are people have long since moved on from them.

I came across the need for more than 1000 partitions many times or this good forum.

A few thoughts on the 15K partition support:

1. I am wondering when will it be available in SQL Server 2012 R2? It is stated that an earlier version (2008 R2) is currently more advanced than the server version (2012 R2), yet most of the current support is not possible from 2008 R2 to 15K support is used.

2. What’s the deal with _db_partitioned_ partitions? I thought that starting 2005, all changes will be made using ALTER commands.

3. Why isn’t replication supported with 15K support? Just like bag shipping and running, I think it could have been supported if subscribers are SQL as well.

Can anyone shed some light here?
What Happens When…

• You need to partition data by day for 3 years of data or more? Or you need to partition data by hour for a year’s worth of data?
  • In SQL Server 2008 & SQL Server 2008 R2
    • Limited to 1,000
      • Unless you installed 2008 SP2 or 2008 R2 SP1 – which allowed for 15,000 partitions when enabled via sp_db_increased_partitions
        • This prevented moving from 2008 SP2 to 2008 R2 RTM
        • Also prevented moving SQL Server 2008 SP2 database with 15,000 partitions enabled to SQL Server 2008 or 2008 SP1
        • Created other restrictions for Log Shipping, Database Mirroring, Replication, SSMS manageability
What Happens When...

• You need to partition data by day for 3 years of data or more? Or you need to partition data by hour for a year’s worth of data?
  • In SQL Server 2012
    • 15,000 partitions are supported in RTM (no SP required)

15,000 Partitions

• You now have the option – as appropriate
  • Flexibility to partition based on common data warehousing increments (hours / days / months) without hitting the limit
    • This doesn’t remove the need for an archiving strategy or mindful planning
  • You have native support for log shipping, availability groups, database mirroring, replication and SSMS management
• Exceptions:
  • > 1000 partitions for x86 is permitted but not supported
  • > 1000 partitions for non-aligned indexes is permitted but not supported
  • For both exceptions – the risk is in degraded performance and insufficient memory

What Happens When...

• Your partitioned index is rebuilt or created:
  • In SQL Server 2008 and SQL Server 2008 R2
    • All table rows are scanned in order to create the statistics histogram
  • In SQL Server 2012
    • A default sampling algorithm is used instead
      • May or may not have an impact on performance
      • You can still choose to scan all rows by using CREATE STATISTICS or UPDATE STATISTICS with FULLSCAN
What Happens When...

2008 ->

2012 ->

Enhanced Manage Partition Wizard

- SQL Server 2008 R2
- SQL Server 2012
Manage Partition Wizard

Demo: Partitioning Enhancements
Summary

- SQL Server 2012 offers
  - An increased number of partitions, helping address common data warehouse requirements
  - Prevention of lock starvation during SWITCH operations
  - Reduced statistics generation footprint (not scanning ALL rows by default)
  - An enhanced manageability experience, enabling wizard-based SWITCH IN and SWITCH OUT assistance

TSQL performance enhancements
OVER Clause Windowing

- Some existing queries do not optimize well
- Example: Details of orders and days since previous order of each product

```sql
-- Traditional approach
SELECT rs.ProductKey, rs.OrderDateKey, rs.SalesOrderNumber,
     rs.OrderDateKey - (SELECT TOP(1) prev.OrderDateKey
                        FROM dbo.FactResellerSales AS prev
                        WHERE rs.ProductKey = prev.ProductKey
                        AND prev.OrderDateKey <= rs.OrderDateKey
                        AND prev.SalesOrderNumber < rs.SalesOrderNumber
                        ORDER BY prev.OrderDateKey DESC, prev.SalesOrderNumber DESC)
     AS DaysSincePrevOrder
FROM dbo.FactResellerSales AS rs
ORDER BY rs.ProductKey, rs.OrderDateKey, rs.SalesOrderNumber;
```
OVER Clause Windowing

-- Windowed approach

```sql
SELECT ProductKey, OrderDateKey, SalesOrderNumber,
OrderDateKey - LAG(OrderDateKey)
OVER (PARTITION BY ProductKey
    ORDER BY OrderDateKey, SalesOrderNumber)
AS DaysSincePrevOrder
FROM dbo.FactResellerSales AS rs
ORDER BY ProductKey, OrderDateKey, SalesOrderNumber;
```
Demo:
New Window
Functions
Performance

Sequences

```
CREATE SEQUENCE Booking.BookingID AS INT
START WITH 20001
INCREMENT BY 10;

CREATE TABLE Booking.FlightBooking
(FlightBookingID INT PRIMARY KEY CLUSTERED
DEFAULT (NEXT VALUE FOR Booking.BookingID),
...)
```
Sequence: Cache or No Cache

• Cache can increase performance for applications that use sequence objects by minimizing the number of disk IOs that are required to generate sequence numbers.

Cache Example

• If the Database Engine is stopped after you use 22 numbers, the next intended sequence number in memory (23) is written to the system tables, replacing the previously stored number.
• If the Database Engine stops abnormally for an event such as a power failure, the sequence restarts with the number read from system tables (39). Any sequence numbers allocated to memory (but never requested by a user or application) are lost. This functionality may leave gaps, but guarantees that the same value will never be issued two times for a single sequence object unless it is defined as CYCLE or is manually restarted.
Cache default?

- If the cache option is enabled without specifying a cache size, the Database Engine will select a size.
- “Don’t count on it being consistent. Microsoft might change the method of calculating the cache size without notice.”
Distributed Replay

Improved Benchmarking and Testing

- Benchmarking and testing are improved through implementation of:
  - Distributed Replay Controller
  - Support for multiple Distributed Replay Clients
What happens when...

- The business needs to perform application compatibility testing prior to performing an upgrade, performance debugging of a highly concurrent workload, system capacity planning, or benchmark analysis of a database workload
  - In SQL Server 2008
    - SQL Server Profiler may be used to replay a captured trace against an upgraded test environment from a single computer
    - Event replay does not follow original query rates

What happens when...

- In SQL Server 2012
  - Distributed Replay can be used to replay a workload from multiple computers and better simulate a mission-critical workload
  - Replay can be configured to reproduce original query rates, or to run in stress test mode where the rate of replay occurs faster than the original query rate
Distributed Replay Components

- **Administration Tool**
  - Command line application that talks to the Replay Controller
- **Replay Controller**
  - Computer running the "SQL Server Distributed Replay Controller" service which is used to control the Replay Clients
- **Replay Clients**
  - One or more computers which run the "SQL Server Distributed Replay Client" service
- **Target Server**
  - SQL Server instance that the replay is directed towards

Distributed Replay Process

- **Event Capture**
  - Events are captured using SQL Server Profiler or a server side trace based on the Replay Trace template
- **Preprocessing**
  - Trace data is parsed into an intermediate file for replay
  - Specifies whether system session activities are included in the replay and the max idle time setting
- **Event Replay**
  - Intermediate file divided among the replay clients
  - After clients receive dispatch data, controller launches and synchronizes the replay operation
  - Each client can record the replay activity to a local result trace file
### Distributed Replay Sequencing Modes

**Synchronization mode**
(Application compatibility and performance testing)
- Events are replayed in the order in which they were submitted during the capture, within and across connections based on the events timestamp
- The replay engine will try to emulate the original query rate observed during the capture

**Stress mode**
(Stress testing and capacity planning or forecasting)
- No order or time synchronization across clients
- Submit order is only maintained within each connection allowing the replay engine to drive more load against SQL Server than in synchronization mode
- `ConnectTimeScale` and `ThinkTimeScale` parameters adjust the degree of stress during replay
  - Actual connect time is multiplied by `ConnectTimeScale/100` to determine replay connect time
  - Actual think time is multiplied by `ThinkTimeScale/100` to determine replay think time
Summary

• SQL Server 2012 offers better benchmarking and testing
  • Distributed Replay supports multiple replay clients allowing for higher scalability during the replay process
  • Replay operations can match original query rates for more accurate analysis of changes to the environment

Scalability using AlwaysOn
What happens when...

- The business wants to:
  - Make use of the mostly-unused failover server(s) for reporting
  - Against real-time business data

SQL Server 2008 R2 or prior

- Database mirroring required snapshot management of the mirrored databases for reporting purposes
  - Snapshot data does not change requiring a new snapshot to keep data up to date, plus connection migration to the new snapshot
  - Snapshots exist until cleaned up, even after failover occurs
  - Reporting workload can block database mirroring process
- Log shipping using RESTORE ... WITH STANDBY provides near real-time access to business data
  - Log restore operations require exclusive access to the database
In SQL Server 2012

- In SQL Server 2012
  - AlwaysOn Readable Secondaries enable read-only access for offloading reporting workloads
  - Read workload on Readable Secondaries does not interfere with data transfer from primary replica
  - Readable Secondaries can be used for offloading backup operations

Topology Example
**Readable Secondary**

**Client Connectivity**

- Client connection behavior determined by the Availability Group Replica option
  - Replica option determines whether a replica is enabled for read access when in a secondary role and which clients can connect to it
  - Choices are:
    - No connections
    - Only connections specifying `Application Intent=ReadOnly` connection property
    - All connections
- Read-only Routing enables redirection of client connection to new readable secondary after a failover
  - Connection specifies the Availability Group Listener Virtual Name plus `Application Intent=ReadOnly` in the connection string
  - Possible for connections to go to different readable secondaries if available to balance read-only access

**Readable secondary**

**Readonly routing**

- Client connects to the Availability Group Listener virtual name
  - Standard connections are routed to the Primary server for read/write operations
  - ReadOnly connections are routed to a readable secondary based on ReadOnly routing configuration
Query Performance on the Secondary

- Challenges:
  - Query workloads typically require index/column statistics so the query optimizer can formulate an efficient query plan
  - Read-only workloads on a secondary replica may require different statistics than the workload on the primary replica
  - Users cannot create different statistics themselves (secondaries can’t be modified)

- Solution:
  - SQL Server will automatically create required statistics, but store them as temporary statistics in tempdb on the secondary node
  - If different indexes are required by the secondary workload, these must be created on the primary replica so they will be present on the secondaries
    - Care should be taken when creating additional indexes that maintenance overhead does not affect the workload performance on the primary replica

Offloading Backups To a Secondary

- Backups can be done on any replica of a database to offload I/O from primary replica
  - Transaction log backups, plus COPYONLY full backups
- Backup jobs can be configured on all replicas and preferences set so that a job only runs on the preferred replica at that time
  - This means no script/job changes are required after a failover
- Transaction log backups done on all replicas form a single log chain
- Database Recovery Advisor tool helps with restoring backups from multiple Secondaries
Workload Impact on the Secondary

• Read-only workloads on mirror database using traditional database mirroring can block replay of transactions from the principal
• Using Readable Secondaries, the reporting workload uses snapshot isolation to avoid blocking the replay of transactions
  • Snapshot isolation avoids read locks which could block the REDO background thread
  • The REDO thread will never be chosen as the deadlock victim, if a deadlock occurs
• Replaying DDL operations on the secondary may be blocked by schema locks held by long running or complex queries
  • XEvent fires which allows programmatic termination/resumption of reporting
    • sqlserver.lock_redo_blocked event

Summary

• SQL Server 2012 allows more efficient use of IT infrastructure
  • Failover servers are available for read-only workloads
  • Read-only secondaries are updated continuously from the primary without having to disconnect the reporting workload
• SQL Server 2012 can improve performance of workloads
  • Reporting workloads can be offloaded to failover servers, improving performance of the reporting workload and the main workload
  • Backups can be offloaded to failover servers, improving performance of the main workload
DBA: Boost Your Server

Reader Writer Blocking

Tran1 (Update)  Tran2 (Select)

Data Page

X-Lock  Row-1  S-Lock  Blocked
Read Committed Snapshot

- New “flavor” of read committed
  - Turn ON/OFF on a database
- Readers see committed values as of beginning of statement
  - Writers do not block Readers
  - Readers do not block Writers
  - Writers do block writers
- Can greatly reduce locking / deadlocking without changing applications
Lock Escalation

- Converting finer-grain locks to coarse grain locks.
  - Row to Table
  - Page to Table.
- Benefits
  - Reduced locking overhead
  - Reduces Memory requirement
- Triggered when
  - Number of locks acquired on a rowset > 5000
  - Memory pressure
Partitioned Tables and Indexes

- SQL Server 2005 introduced partitioning, which some customers use to scale a query workload
  - Another common use is to streamline maintenance and enable fast range inserts and removals from tables

---

Lock Escalation: The Problem

- Lock escalation on partitioned tables reduces concurrency as the table lock locks ALL partitions
  - Only way to solve this in SQL Server 2005 is to disable lock escalation
Lock Escalation: The Solution

- SQL Server 2008 & up allows lock escalation to the partition level, allowing concurrent access to other partitions.

- Escalation to partition level does not block queries on other partitions.

**Demo:**
Partition Level Lock Escalation
Filtered Indexes

- An index with a WHERE clause to specify a criteria
- Essentially index only a subset of a the table
  - Query optimizer most likely to use clause matches that of the filtered

Using Filtered Indexes

- Advantages of Filtered Indexes
  - Improve query performance, partly by enhancing execution plan quality
  - Smaller index maintenance costs
  - Less disk storage
Using Filtered Indexes

- Scenarios for Filtered Indexes
  - Sparse columns, where most data is null
  - Columns with categories of values
  - Columns with distinct ranges of values

Demo: Filtered Indexes
Optimize for ad hoc workloads

- New server option in SQL Server 2008
- Only a stub is cached on first execution
- Full plan cached after **second** execution

- `SP_CONFIGURE 'show advanced options',1`  
  `RECONFIGURE GO`
- `SP_CONFIGURE 'optimize for ad hoc workloads',1`  
  `RECONFIGURE GO`

How good is it?
Demo: Optimize for ad hoc workload

Configuration: Don’t get confused!
Unhandled Error

Error Details

File
The transaction log for database 'SQLPASS_MAIN' is full. To find out why space in the log cannot be reused, see the log_reuse_wait_desc column in sys.databases

Processor Funny Games
TempDB

- Create a file per CPU that SQL Server uses
- Not more than 8

Memory Configuration

- X86? Really?
  - Microsoft Knowledge Base article 274750
Lock Pages in Memory

- Yes/No?
- Enterprise Edition only
  - Can be done on Standard using latest SPs for 2005/2008 and trace flag 845 for 2008 R2
- AWE is ignored in 64 bit
- The ‘Local System’ account has the ‘lock pages in memory’ privilege by default
- Configure MaxServerMemory

Min/Max Memory

- Yes/No?

<table>
<thead>
<tr>
<th>Physical RAM</th>
<th>MaxServerMem Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>2GB</td>
<td>1500</td>
</tr>
<tr>
<td>4GB</td>
<td>3200</td>
</tr>
<tr>
<td>6GB</td>
<td>4800</td>
</tr>
<tr>
<td>8GB</td>
<td>6400</td>
</tr>
<tr>
<td>16GB</td>
<td>10000</td>
</tr>
<tr>
<td>32GB</td>
<td>13500</td>
</tr>
<tr>
<td>64GB</td>
<td>21500</td>
</tr>
<tr>
<td>128GB</td>
<td>26000</td>
</tr>
<tr>
<td>256GB</td>
<td>44000</td>
</tr>
<tr>
<td>512GB</td>
<td>68000</td>
</tr>
<tr>
<td>1TB</td>
<td>92000</td>
</tr>
<tr>
<td>2TB</td>
<td>124000</td>
</tr>
</tbody>
</table>
### Min/Max Memory

<table>
<thead>
<tr>
<th>Physical RAM</th>
<th>Target Avail RAM in Task Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 4GB</td>
<td>512MB – 1GB</td>
</tr>
<tr>
<td>4-32GB</td>
<td>1GB – 2GB</td>
</tr>
<tr>
<td>32-128GB</td>
<td>2GB – 4GB</td>
</tr>
<tr>
<td>&gt; 128GB</td>
<td>&gt; 4GB</td>
</tr>
</tbody>
</table>

- Yes/No?
- Keep an eye on Available MB counter
Reason and Solution?

- Note the name **Memory (Private Working Set)** –
  - AWE APIs are used on 64bit to “lock” pages
  - That memory is not part of the working set
- Only trust:
  - Perfmon SQL Server memory counters
  - sys.dm_os_process_memory DMV
Demo: Lock Pages in Memory

Compression
As data volume grows...

- Large databases =
  - Storage Cost
  - Workload Performance
  - Manageability Cost
  - Backup/Recovery

Compression

- Store data efficiently in the row/page
  - (+) More data can fit in memory
  - (+) Better Performance for I/O bound workload
  - (-) Performance degradation for CPU bound workload
**Data Compression**

- SQL Server 2008
  - ROW and PAGE compression
  - Backup Compression
- SQL Server 2008 R2
  - Unicode compression
- SQL Server 2008 R2
  - Spatial indexes compression

**Enabling Compression: Microsoft’s Perspective**

- Latest partition uncompressed
  
<table>
<thead>
<tr>
<th></th>
<th>Jan-Mar</th>
<th>Apr-June</th>
<th>July-Sept</th>
<th>Oct-Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
</tr>
<tr>
<td>PAGE Compressed</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>ROW Compressed</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>
Should it be so complex?

In real life – usually compress the entire large tables using page compression...

Summary – Compression

• Can reduce size of database significantly
• Lower total cost of ownership (TCO)
• Easy to enable/disable
• No application changes
• Performance gains!
Demo: Compression

see you
Next Year
at SQL Explore