**(Get Dynamic Management Views/Functions)**

select name, type\_desc from sys.all\_objects where name like 'dm%' order by name

**OR**

SELECT \* FROM sys.all\_objects WHERE [name] LIKE '%dm\_%' AND [type] IN ('V', 'TF', 'IF') AND [schema\_id] = 4 ORDER BY [name]

**AND (Get DataBase Console Commands)**

DBCC HELP ('?') -> DBCC HELP (Some DBCC like CHECKDB)

**AND (Get version/server properties)**

SELECT @@VERSION

**OR**

SELECT SERVERPROPERTY('Edition') AS Edition, ...ProductLevel, ProductVersion, etc

**AND (Get system stored procedures)**

SELECT name FROM sys.all\_objects WHERE type = 'P' AND is\_ms\_shipped = 1

**AND (Get schema and table names for db)**

SELECT \* FROM sys.(tables/columns/views/etc)

**AND (Get info on physical drive data that SQL can use)**

EXEC master..xp\_fixeddrives

**DAC (Dedicated Administrator Connection) port 1434**

* SQLCMD -A -S server\instance
* File->New->Db Engine Query->ADMIN:Instance

**Work examples:**

* Cut downtime for forklifts by pre-restoring SQL databases and rolling logs forward
* Complete SQL Server recovery of payroll (the day before payday!), corrupt master which turned out to be corrupt resource database -> uninstall SQL Server reinstall, patch, restore master and msdb and then app db's; 17 hours straight Microsoft Sev A
* VLF issue causing failed upgrade on physical server, memory error
* Created AMPFM web reports app in 2 weeks (created in 2004 and still in use)
* My own SQL daily checks data collection and presentation
* Cannibalize Microsoft Assessment and Planning toolkit in combination with some SCOM tables and Powershell to create a new and improved SQL monitor/daily checks database possibly using Power View for some pretty dashboards and SSRS for reports & auto nightly db restores
* Business Intelligence and analytics

**My troubleshooting process:**

* Low hanging fruit: SQL Log, Event Viewer\cluster logs, Activity Monitor (Dev SSMS sessions, blocking/locks, poor performing queries)
* Deeper: Reports (index fragmentation, poor performing queries, blown stats), sp\_who(2), Windows Perfmon (resource bottlenecks: disk/cpu/memory), Server Dashboard
* Really deep: DMV's, DBCC's, Profiler trace/Extended events, Server side trace (off hours random events), MS process explorer utility, my bag of tricks (custom queries collected/written over time)

**Daily Checks:**

* Instances up and running, Failed jobs, Failed backups, Errors in SQL Logs**,** Unusual growth (data, log, tempdb), Unusual performance (resource bottlenecks, blown stats), Rejected login attempts, AlwaysOn Synchronization status, Integrity check results, Check nightly database restores
  + My homegrown webpage/scripts
  + Emails from custom alerts
  + SCOM
  + Avamar
  + Help ticket system (Service Pro)
  + Help Desk Turnover Log (Which I wrote also 10+ years ago!)

**Index Related**

* sys.dm\_db\_index\_usage\_stats, sys.dm\_db\_index\_physical\_stats

**Execution Related**

* sys.dm\_exec\_/connections/sessions/requests/cached\_plans/query\_plans/sql\_text/query\_stats

**SQL OS**

* sys.dm\_os\_wait\_stats - What are we waiting on?

**I/O**

* sys.dm\_io\_virtual\_file\_stats - look at the IO stalls for both reads and writes, could be storage issue

**Traces -**

* **SQL Profiler** (old, not being developed, can be scheduled to stop, can see perfmon counters side by side)
* **Extended Events** (best in 2012+ now wi. GUI, results can be aggregated, low impact, new features; AlwaysOn, InMem stuff, etc...only covered using EE)
* **Server Side Trace -** sp\_trace\_create; You can script out a trace which gives you greater flexibility than having to use profiler.

**Windows Performance Monitor;**

**Workload (SQL Server: Statistics)=**

* *SQL Compilations/sec*<10% of batch requests/sec-Lower is better, could be too much ad-hoc querying consider stored procs
* *Recompilations/sec*<10% of comps/sec-Lower is better near 0 is best
* SQL Attention Rates/sec ~0-Number of cancels and query timeouts

**Memory (SQL Server: Buffer Manager)=**

* *Page Life Expectancy*>300-Life of data page in cache; high is good, mem pressure
* Page Reads&Writes/sec<90-higher may mean poor indexing or memory pressure
* (Page Lookups/sec) / (Batch Requests/sec)<100-Inefficient execution plans

**Users & Lock performance (SQL Server: Latches)=**

* Lock Waits/sec=0-Above 0 indicates blocking
* *Lock requests/sec*<1000-Higher numbers indicate queries are accessing large data sets and may need some tuning

**Data Access (SQL Server: Access Methods)=**

* *Full Scans/sec* – High numbers could be caused by missing indexes, a sudden jump could indicate blown statistics (Index searches/sec 1 Full scan per 1k index searches)
* Page Splits/sec <20 per 100 Batch Requests/sec-High numbers indicate overflowing index pages, may indicate that fillfactor needs to be adjusted

**CPU & Processor=**

* %Processor Time (sqlserver) <80%-Higher could indicate processor pressure
* Processor Queue Length <4 per cpu- <12 good, <8 better, <4 best
* Context Switches/sec <1500 - >6000 poor, <3000 good, <1500 excellent

**SQLCMD -** command line sql (-A DAC, -E trusted connection, -S server\instance, -U; -P user/pass)

**Storage layout; SAN level/data file configuration (growth)/TempDB**

* Set partitions (cluster size) to 64k allocation units to match SQL’s extent size for both SSD and spinning disk
* Compression=less to read off disk (IO most expensive), incurs some overhead at time it is read in memory
* TempDb - 1 data file per core up to 8, fixed size
* RAID levels- RAID 5 for the databases; and RAID 10 for the transaction logs and tempdb
* SSD consideration - Throughput: Server has two PCI Express 3.0 (x1 Link) slots on the motherboard. Each slot is capable of up to 7.69 gigabits per second. However, if both slots are on the same back-end bus and you install one HBA in each slot, the total amount of bandwidth available to both cards will be only 7.69 gigabits.

**SQL Instance/database/server configuration; memory/MDOP/Lock Pages in Memory (defunct in newer versions of SQL and with VM's?)/ Trace Flag considerations**

**sp\_configure=**Displays or changes global configuration settings for the current server (MDOP, Backup Compression, Mem settings Min/Max, remote access, xp\_cmdshell<-biggy)

* 'show advanced options' 1
* RECONFIGURE (if allowed for that setting aka dynamic)

**DB Config**

* auto-stats, auto-close, auto-shrink
* You can also enable certain query configs here
* Parameterization: Simple (default), Forced - SQL will attempt to turn literals into parameters in an attempt to create better reusable query plans

**SQL Server 'Resource' database:** contains all system objects that are included with SQL Server

**SQL Server Connection Pooling**

* In order for connections to pool they must have the same connection properties
* 100 connections per pool (default)
* Connection leakage - Happens when db connections are not properly closed by the app
  + Timeout default 15 seconds
* Pool fragmentation -
  + Integrated security results in unique connections=0ne pool per user, instead use fewer integrated security accounts (I’ve often seen 1 generic for all app connections) or better use windows authentication for increased security
  + Instance has many databases and each is being connected to individually, instead use a ‘USE’ in queries to switch between databases

**Transactions**

* Autocommit transactions - Each individual statement is a transaction.
* Explicit transactions - Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.
* Implicit transactions - A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement.
* Batch-scoped transactions - Applicable only to multiple active result sets (MARS), a Transact-SQL explicit or implicit transaction that starts under a MARS session becomes a batch-scoped transaction. A batch-scoped transaction that is not committed or rolled back when a batch completes is automatically rolled back by SQL Server.
* At the beginning of stored procedure one should put [SET XACT\_ABORT ON](http://msdn.microsoft.com/en-us/library/aa259192%28v=sql.80%29.aspx) to instruct Sql Server to automatically rollback transaction in case of error. If omitted or set to OFF one needs to test [@@ERROR](http://msdn.microsoft.com/en-us/library/aa933181%28v=sql.80%29.aspx) after each statement or use [TRY ... CATCH rollback](http://msdn.microsoft.com/en-us/library/ms175976.aspx) block
* RAISERROR or THROW to write some custom error or message

**Lifecycle of a SQL Server request**

Request is submitted generating a task -> task is picked up by a worker -> parse request, check plan cache for existing plan if one does not exist then generate a new plan (query optimizer) and store it in the plan cache -> execute query plan, check buffer cache to see if data is in memory if not then go to disk to fetch data into buffer cache (buffer manager) -> Release worker and return results

* DBCC FREEPROCCACHE - clear the entire or just specific plans from the plan cache
* DBCC DROPCLEANBUFFERS - write dirty pages to disk then clear the cache

**The correct order of the Logical Query Processing Phases is as follows:**

1. FROM

2. ON

3. OUTER

4. WHERE

5. GROUP BY

6. CUBE | ROLLUP

7. HAVING

8. SELECT

9. DISTINCT

10. TOP

11. ORDER BY

**Stored Procedure:**

A stored procedure is a named group of SQL statements that have been previously created and stored in the server database. Stored procedures accept input parameters so that a single procedure can be used over the network by several clients using different input data. And when the procedure is modified, all clients automatically get the new version. Stored procedures reduce network traffic and improve performance. Stored procedures can be used to help ensure the integrity of the database.

**Advantages of Using Stored Procedures:**

* Stored procedure can reduce network traffic and latency, boosting application performance.
* Stored procedure execution plans can be reused; they stay cached in SQL Server’s memory, reducing server overhead.
* Stored procedures help promote code reuse.
* Stored procedures can encapsulate logic. You can change stored procedure code without affecting clients.
* Stored procedures provide better security to your data.

**Triggers:**

A trigger is a SQL procedure that initiates an action when an event (INSERT, DELETE or UPDATE) occurs. Triggers are stored in and managed by the DBMS. Triggers are used to maintain the referential integrity of data by changing the data in a systematic fashion.

**There are two types of Triggers.**

1) DML Trigger

There are two types of DML Triggers

1.Instead of Trigger

Instead of Triggers are fired in place of the triggering action such as an insert, update, or delete.

2. After Trigger

After triggers execute following the triggering action, such as an insert, update, or delete.

2) DDL Trigger

This type of trigger is fired against Drop Table, Create Table, Alter Table or Login events. DDL Triggers are always After Triggers.

**User-defined Functions (UDF):**

User-defined Functions allow defining its own T-SQL functions that can accept zero or more parameters and return a single scalar data value or a table data type.

Different Types of User-Defined Functions created are as follows:

**Scalar User-defined Function**

A scalar user-defined function returns one of the scalar data types. Text, ntext, image and timestamp data types are not supported. These are the type of user-defined functions that most developers are used to in other programming languages.

**Inline Table-Value User-defined Function**

An Inline table-value user-defined function returns a table data type and is an exceptional alternative to a view as the user-defined function can pass parameters into a T-SQL select command and in essence provide us with a parameterized, non-updateable view of the underlying tables.

**Multi-Statement Table-Value User-defined Function**

A multi-statement table-value user-defined function returns a table, and it is also an exceptional alternative to a view as the function can support multiple T-SQL statements to build the final result where the view is limited to a single SELECT statement. Also, the ability to pass parameters into a T-SQL select command or a group of them gives us the capability to in essence create a parameterized, non-updateable view of the data in the underlying tables. Within the create function command, you must define the table structure that is being returned. After creating this type of user-defined function, It can be used in the FROM clause of a T-SQL command unlike the behavior encountered while using a stored procedure which can also return record sets.

**Aggregate Functions:**

Aggregate functions perform a calculation on a set of values and return a single value. Aggregate functions ignore NULL values except COUNT function. HAVING clause is used, along with GROUP BY for filtering query using aggregate values.

The following functions are aggregate functions.

AVG, MIN, CHECKSUM\_AGG, SUM, COUNT, STDEV, COUNT\_BIG, STDEVP, GROUPING, VAR, MAX, VARP

**What is the Difference between a Function and a Stored Procedure?**

UDF can be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section, whereas Stored procedures cannot be. UDFs that return tables can be treated as another rowset. This can be used in JOINs with other tables. Inline UDF’s can be thought of as views that take parameters and can be used in JOINs and other Rowset operations.

**Query Execution Memory Grant; aka query work buffer**

* Used to store temporary row data while sorting and joining rows
* Estimate generated at the execution of a request of the memory cost for that request
* sys.dm\_exec\_query\_memory\_grants - This DMV shows all queries consuming memory grants including those waiting in Resource Semaphore queue.
* sys.dm\_os\_wait\_stats - This DMV shows wait statistics of all server objects. Memory grant uses "RESOURCE\_SEMAPHORE" wait type. If you see significant waits on this wait type, you may have an issue with big queries.
* DOP (Parallel processing uses more memory and workers, each thread gets its own set)
* Memory pressure can lead to tempdb use which will be SLOW
* SORT warnings - large sorts may result in memory overflow of the memory grant space which leads to TempDb usage = SLOW query

**SQL query optimization:**

* Treating data as sets instead of rows (encyclopedia set example->Books A-Z->Index A-Z)
  + Instead of ‘IN’ use ‘EXISTS’ - Again treat your data as a set not on a row by row basis
* Query Execution Plans (Expected vs. Actual) - A road map for how the query optimizer is deciding how to process a query.
  + SHOWPLAN\_TEXT, SHOWPLAN\_ALL, Graphical Query Plan, sys.dm\_exec\_query\_stats
* Avoid using functions in the predicate/where clause, will invalidate the use of any potential indexes
* SET STATISTICS IO ON/SET STATISTICS TIME (scan count, # logical/phys reads)
* sys.dm\_exec\_sql\_text - This function is used to return sql text from a batch by passing in the sql\_handle.
  + FROM sys.dm\_exec\_query\_stats AS qs CROSS APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) AS st
  + FROM sys.dm\_exec\_requests r CROSS APPLY sys.dm\_exec\_query\_plan(plan\_handle) as qp CROSS APPLY sys.dm\_exec\_sql\_text(r.sql\_handle)

**Set Operations:**

* **Union/Union All**=Concats data sets, eliminates duplicates/does not eliminate duplicates
* **Intersect**=Shows only rows present in both sets
* **Except**=Shows only rows that do not appear in second set

**Buffer Cache/pool -** memory area used for holding data pages

* many buffers; 8kb ea, this is where the stuff read from disk goes
* LRU (Least Recently Used) - keep your important data in memory!
* CHECKPOINT, DBCC DROPCLEANBUFFERS - write dirty pages to disk then clear the cache
* Lazy Writer Process - Periodically comes along to push dirty pages down to disk and frees up buffer cache
* sys.dm\_os\_buffer\_descriptors - one row for each data page in the buffer cache
* DBCC MEMORYSTATUS - Shows how the buffer cache is divided up, mem usage, buffer distribution
* May need to run DBCC TRACEON(3604) first.
  + DBCC CACHESTATS - Info about objects in buffer cache, Displays pages used by object plans, SQL plans, temp tables,

**Buffer Manager -**

* Interface between the database engine and the in-memory pages
* Hash tables - bucketed collections of pointers to each buffer in specific memory locations aka the buffer array and the pages written to each buffer

**Right sizing databases (default settings are bad)/vlf/fragmentation**

* I/O stats for db files including logs; sys.fn\_virtualfilestats (old)/sys.dm\_io\_virtual\_file\_stats(new)
* Data File-Page/8k - Header = first 96 bytes
  + Smallest storage unit in SQL Server: Important because only whole pages are loaded into memory this is where have a low fill factor can waste resources
* Extent-8 (physically contiguous) pages/64k - (FYI 128 pages = 1MB)
  + Uniform Extent - All 8 pages belong to 1 table
  + Mixed Extent - 8 pages are shared by 2 or more tables
* TLog-log record/variable size
* GAM (uniform extents; track 64k extents or up to about 4GB) & SGAM (mixed extents; 4GB) are used to track space allocation along with IAM (extents for HEAPS (tables), LOBs, Indexes and partitions; 4GB) These are SQL’s map to the data
  + DBCC TRACEON (3604); -> DBCC PAGE - displays contents of data page (extremely low level troubleshooting stuff!)

**T-Logs -**

* Log cache - For each logical write, a transaction log record is inserted in the log cache that records the modification. The stuff in log cache must be written to disk before the associated dirty page is written to disk and removed from the buffer.
* Stores DML and DDL queries
* Sequential writes
* Can have more than 1, but why? No performance increase because they are written to sequentially
* Right size it up front to prevent excessive vlf’s; High numbers of vlf’s can affect db recovery times (I have seen out of memory errors caused by excessive vlf’s)

**TEMPDB -**

* When TempDB starts getting full, it’s an indicator that there’s a lot of temporary object creation as well as out of memory sorting and joining going on in the database. None of this is necessarily bad, but they should be checked up on.

**SQL Page Types**

Data, Index, Text/Image, GAM (global allocation map), SGAM (Shared),IAM (Index allocation map), PFS (Page Free Space), BCM (Bulk Change Map), DCM (Differential Change Map)

**Latches - An object that ensures data integrity on other objects in SQL Server memory, particularly pages, controlled by the SQL OS, (locks can be manipulated by setting isolation level, latches cannot)**

* Buffer Latch - Writing to a page in memory must be a single threaded process; wait your turn buddy! Prevents ‘lost updates’ = one thread overwriting another threads change
* IO Latch - Latches also make sure that duplicate data is not brought into the buffer pool from disk and that in use data is not prematurely flushed from the buffer pool to disk.
* Is very short (normally)
* Buffer/IO latches - sys.dm\_os\_waitstats
* Possible causes of too much latching;
  + A high number of updates to a small number of records
  + Competing parallel processing of transactions
  + Extremely (ridiculously) high numbers of inserts on tables that are also being read (something like a monitoring app that is collecting data every few seconds)

**Waits**

* cxpacket - Experienced in parallel processing when a thread on one cpu finishes its task and has to wait for one or more others to finish in order to proceed

**Lock Types - Logical construct that ensures controlled access to a resource and isolationism when required for pages in use, (locks can be manipulated by setting isolation level, latches cannot)**

* Lasts the duration of transaction
* Shared - Used for read queries that don't include any DML
* Update - Used on updateable resources to help prevent deadlocks
* Exclusive - Used for DML SQL queries, tries to ensure multiple changes aren't being made to data at the same time
* Schema - Used for things like Table changes, DDL queries, like adding a column
* Intent - Used to establish lock hierarchy
  + shared, exclusive, shared exclusive
* Bulk Update
* Granularity
  + Row->Key->Page->Extent->HeapOrBTree->Table->File->Database

**Deadlocks:**

* When two or more processes permanently block each other by each having a lock on a resource the other process needs
* Resources that can lead to deadlocks: locks, worker threads, memory
* Deadlock monitor - Will pick a victim and kill their transaction to free up the deadlock
* High noon gunslinger standoff!

**Isolation Levels - (locks can be manipulated by setting isolation level, latches cannot)**

* **Serializable -** Cleanest reads possible, but also causes the most locking
  + Lock held for the entire duration of the transaction
* **Committed read**=Locking read sees each row as it was at the time it was briefly locked and physically read, it is possible to see phantom data. You may not be seeing the most recent updated data.
  + Locks held for short period, only long enough to process each row.
* **RCSI**=sees all rows as they were at the time the statement began
  + Once snapshot isolation is enabled, updated row versions for each transaction are maintained in tempdb.
* **Uncommitted Read/NOLOCK**=Reads data while it is in possible transition, could be changed or deleted, could also find more than one version of the data. Dirty reads, Missing rows, read rows twice, read multiple versions of same row, read error

**SQL Server Backups\Restore:**

* Modes: Full, Simple, Bulk (minimal logging, no point in time recovery for bulk insert period)
* Types: Full, Differential, Copy Only (does not break diff chain), TLog
* Restore VerifyOnly
* msdb: backupset, backupfile, restorefile, restorehistory
* Take a tail log backup if your db crashes and you want to be able to restore it to that point

**SQL Server Restore phases:**

* Analysis/Data Copy - Move data from full/diff backup into data files
* Redo - Apply log backups to roll the data forward in time
* Undo - Any outstanding transactions that may have been only partly run are evaluated and either rolled forward or backward in order to make the database transactionally consistent.

**Error trapping in SQL Queries**

* TRY/CATCH

**Index creation/usage/fragmentation/Fill Factor settings/statistics/heap (nonclustered table, no order)/B-Tree (1-200=1-100&101-200=1-50&50-100&101-150&150-200 etc…**

* Tables without a clustered index are 'heaps', fillfactor does not apply
* Having a Fill Factor of 100 (default) is better for tables with little change, a lower fill factor will be more beneficial for tables that have a high rate of change as indexes will fragment slower, tweak it down as necessary...too low can be bad too (wastes space, create overhead)
* Clustered=1 per table because the db physically sorts the table that way, not a different object like other indexes. Getting to the correct leaf will contain the actual row.
  + Narrow – as narrow as possible, in terms of the number of bytes it stores, identity int vs. GUID
  + Unique – to avoid the need for SQL Server to add a "uniqueifier" to duplicate key values
  + Static – ideally, never updated
  + Ever-increasing – to avoid fragmentation and improve write performance

ex. Since query plan generation looks at the selectivity of only the first column in a multi-column index, make your most likely WHERE/JOIN candidates first

* Non-Clustered=Getting to the correct leaf will point to the clustered table or heap
* Columnstore= Compress the data down in a column until you only have unique data and then store each column in its own page, will only need to look at the reduced size pages holding the columns being queried, keeps some kind of master hash to reassemble whole rows
  + Greatly improved and more flexible in 2014 & 2016, not just for dw anymore?
* Indexed Views=can be a great way to speed up a query on an already created data set. Can’t reference other views, or use count, min, max, top or outer joins. Not only can you have a unique clustered index (materialized), it is a requirement for creating other indexes!
  + Indexed views are best built on relatively static underlying tables, otherwise changes to the base tables can result in necessary updates to the materialized view = overhead.
  + Simply put, if we do not use the WITH (NOEXPAND) hint (always in SQL STANDARD, sometimes in ENTERPRISE) when querying an indexed view, the query optimizer will not use statistics created on the indexed view and neither will it create or update statistics automatically
  + the need to create them with the SCHEMABINDING option, use fully qualified table references, and use COUNT\_ BIG ( \*) if the view definition contains a GROUP BY clause.
  + Another, implied but not discussed directly, is that the indexed view definition can only reference tables, not other views.

**Statistics**: 2 kinds; Column and Index

* SQL Server Query Optimizer uses statistics to estimate the distribution of values in one or more columns of a table or index views, and the number of rows to create a high-quality query execution plan.
* Often columns being used in JOIN, WHERE, ORDER BY, or GROUP clauses are good candidate to have up-to-date statistics on them.
* The SQL Server Query Optimizer creates single column statistics when the AUTO\_CREATE\_STATISTICS database property is set to ON or when you create indexes on the table or views (statistics are created on the key columns of the indexes)
* There might be times when you need to create additional statistics using the CREATE STATISTICS command to capture cardinality, statistical correlations so that it enables the SQL Server Query Optimizer to create improved query plans.
* Sampling versus 'With Fullscan' stats
* DBCC SHOWSTATISTICS/sys.dm\_db\_stats\_properties

**Index maintenance**

* DBCC SHOWCONTIG WITH FAST, ALL\_INDEXES, TABLERESULTS
* Alter Index:Reorganize/Rebuild OR DBCC INDEXDEFRAG/DBREINDEX
* Reorg or Rebuild -
  + > 10 percent and < = 30-40 percent ALTER INDEX REORGANIZE
  + > 30-40 percent ALTER INDEX REBUILD
* Differences
  + Rebuild updates index stats with fullscan (but not column stats), reorg does not
  + Reorgs on online operations by default, rebuilds are not; but some can be using ONLINE=ON
  + Reorgs move things around inside of the existing pages, rebuild uses new pages
  + Reorg can be stopped part way and any gains are kept, rebuild is transactional making it all or nothing
* Create Index...Drop Existing or Drop Index; Create Index

**Stored Procedure Recompilation; Pre-run SP's**

* To create a new query plan associated with the SP, maybe the old one is bad
* DBCC FREEPROCCACHE - clear the entire or just specific plans from the plan cache
* You can put a WITH RECOMPILE in the SP create statement to force a new plan each time it is run but this will take more CPU...is it worth it?
* Parameter sniffing - sometimes a plan gets generated using one set of parameters and another set comes along and the plan does not work; maybe using WITH RECOMPILE would work here
* Might be good after a stats refresh
* You can pre-populate your plan cache with some of the most used query plans after a SQL Server restart before it affects any user's

**SQL Server Virtualization considerations: memory ballooning/NUMA (non uniform memory access)/Hyperthreading**

* When setting the SQL Server Lock Pages in Memory user right, the virtual machine’s memory reservation should also be set to match the amount of the provision memory.
* Setting virtual machine memory reservations prevent the balloon driver from inflating into the SQL Server virtual machine’s memory space.

**Integrity Checks -** Over 99.99% (by most accounts) the root cause of corruption is due to problems at the IO subsystem level

* DBCC CHECK/DB/TABLE/FILEGROUP
* Do not reboot, Do not detach db's, Evaluate what type of corruption (metadata issue aka referential integrity error versus bad disk sectors or the like)
  + If metadata issue try DBCC UPDATEUSAGE or I have simply dropped and recreated suspect indexes (if you're lucky and it is an easy object to replace!)
  + If other it's a tougher road maybe 'DBCC CHECKDB dbname WITH REPAIR\_REBUILD'
  + Restore the database from a good backup either as a whole or in parts, aka filegroups
  + Or if all else is lost and as an absolute last resort the dreaded DBCC CHECKDB and REPAIR\_ALLOW\_DATA\_LOSS, and be ready to find a new job…
* For TLog loss
  + ALTER DATABASE <dbname> REBUILD LOG ON NAME = <dbname>, FILENAME = '<logfilepath>')
  + Restore from a good set of backups
  + or the old hail mary REPAIR\_ALLOW\_DATA\_LOSS will rebuild the log
* Use AlwaysOn! Automatic page repair from secondary

**AlwaysOn:**

* dm\_hadr\_: availbility\_group\_states, availability\_replica\_states, cluster, listener state
* Max latency (default) The primary will wait 10 seconds without ACKs. This is configurable via SESSION\_TIMEOUT option
  + Right click AG -> properties
* Failover Cluster Manager -> Roles -> AG -> Properties
  + Lease Timeout
  + HealthCheck Timeout
  + Verbose Logging
* Extended event logs - The sp\_server\_diagnostics results are stored in the clustered diagnostic log files in the SQL Server \LOG directory with file names SRVNAME\_SQLINSTANCENAME\_SQLDIAG\_XXX.XEL.
* DROP/CREATE/ALTER Avail Group
* Flexible failover policy
* connection routing table

**Contained Databases:** Wraps up some of the instance specific stuff (security being a biggie) to send along with the database.

**Cardinality -**

* Thinking mathematically, it is the number of elements in a set (like for figuring out statistics).
* Thinking in the database world, cardinality has to do with the counts in a relationship, one-to-one, one-to-many, or many-to-many (this is what we would use for database design)

**What is RDBMS?**

Relational DataBase Management Systems (RDBMS) are database management systems that maintain data records and indices in tables. Relationships may be created and maintained across and among the data and tables. In a relational database, relationships between data items are expressed by means of tables. Interdependencies among these tables are expressed by data values rather than by pointers. This allows a high degree of data independence. An RDBMS has the capability to recombine the data items from different files, providing powerful tools for data usage.

**What are the Properties of the Relational Tables?**

Relational tables have the following six properties:

* Values are atomic.
* Column values are of the same kind.
* Each row is unique.
* The sequence of columns is insignificant.
* The sequence of rows is insignificant.
* Each column must have a unique name.

**What is Normalization?**

Database normalization is a data design and organization process applied to data structures based on rules that help building relational databases. In relational database design, the process of organizing data to minimize redundancy is called normalization. Normalization usually involves dividing a database into two or more tables and defining relationships between the tables. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

**Normalization: 1NF, 2NF, 3NF, 4NF (EF Codd)**

* **1NF=**Eliminate duplicative columns, create a separate table for each set of related data, identify each row with one or more unique columns as a primary key (address1, address2, address3, etc...also one data item in each column; don't just cram all addresses into one field!)
* **2NF=**Eachnon-keycolumn must be functionally dependent on the entire PK, create separate tables for sets of values that apply to multiple records, relate the tables with a foreign key (break out a list of items from an order table with price and description, etc)
* **3NF=**Eliminate transitive functional dependency; remove anything that does not directly depend on the primary key (Bookid (PK), GenreID, Genre, Price...Genre is related to BookID only through GenreID so should be kicked over into its own table with GenreID tying it back)
* **4NF=**Remove anything derived or calculated from other data; totals and the like
* Reason to denormalize data - possible performance gains by reducing the number of joins required to access data

**ACID**

* **Atomicity-**Database modifications must follow an all or nothing rule. If any part of a transaction fails it is rolled back. Transactions
* **Consistency-**Only valid data will be written to the database. All data entered must obey the rules of consistency or be rolled back. Constraints,
* **Isolation-**Multiple transactions being run at the same time not impact each other's execution. Latches & Locks
* **Durability-**Any transaction that is committed to the database will not be lost. T-Logs

**Constraints**

* NOT NULL - Indicates that a column cannot store NULL value
* UNIQUE - Ensures that each row for a column must have a unique value
* PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
* FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table
* CHECK - Ensures that the value in a column meets a specific condition
* DEFAULT - Specifies a default value when specified none for this column

**Scaling Out SQL Server**

* Linked Servers and distributed queries
* Distributed Partitioned Views / DPV (Transparent scale out) - Uses linked servers to achieve transparency, Reliant upon good partitioning keys, great for OLTP, high management overhead
  + Cross server join queries are bad
  + Enabling Lazy schema validation may help performance - Speed up queries to remote tables by not requesting metadata for each request, only when data is actually needed from the tables
* Data Dependent Routing / DDR (B-TREEesque) - requires a middle Tier to know where to go to get the data, grab a userid and use a lookup table to route to the correct instance, we would also still need to leverage partitioning and we could easily end up in some kind of a hybrid scale-out/scale-up scenario
  + Taking it a step further we can potentially use AlwaysOn most by routing connections via the middle tier/application based on need like where to send a DML versus a read; primary versus a readable secondary
  + Does SQL Browser figure into this?
* How else might we leverage AlwaysOn? Would it help deal with backups and maintaining consistency? Each shard in a silo has an Always On secondary
* Sharding? Is this just partitioning by another name?

**ETL (Extract, Transform, Load) - Primarily used for OLAP/analytics systems we utilize Change Data Capture (CDC) to achieve this.**

**Table Partitioning (Scale Up) is different then Distributed Partition Views (Scale Out)**

* When SQL Server performs data sorting for I/O operations, it sorts the data first by partition. SQL Server accesses one drive at a time, and this might reduce performance.
* In addition, you can improve performance by enabling lock escalation at the partition level instead of a whole table. This can reduce lock contention on the table.
* Partition/Shard types
  + Range: 0-100, 101-200, 201-300; no guarantee of distribution or activity
  + List: Group based on intrinsic data, logical groupings; location or department; uneven growth, can become very complex
  + Hash: Apply a hash function to a key; Round robin data among our shards, can be hard to find data among all the shards

**DBCC commands-**

* CHECKCATALOG – Consistency between tables
* CHECKCONSTRAINTS – For a table
* SQLPERF – T-Log size info
* SHRINKFILE - Shrink a database file
* INPUTBUFFER - used to see last query executed for a particular SPID
* ERRORLOG - cycle SQL log
* LOGINFO - VLF count
* PERFMON - Lists waits, Read/write, worker info, Network info, Buffer Cache/Phys IO
* OPENTRAN - Find zombie process

**Wishlist**

* I wish I could break out and resize the SQL Server activity monitor windows
* I would like to see an option to only run SQL Server Agent jobs when the current instance is the AlwaysOn primary

**Dealing with really large data sets:**

**MultiServer Query:**

SSMS 2008 has a feature to run a query on different servers from one query editor window. First of all, make sure that you registered all the servers under your registered server. Once they are registered, right click on server group name and click New Query.

**Aggregate Tables:**

An aggregate table contains the summary of existing warehouse data, which is grouped to certain levels of dimensions. It is always easy to retrieve data from aggregated tables than visiting original table which has millions of records. Aggregate tables reduce the load in the database server and improve the performance of the query, and they also can retrieve the result quickly.

**Data Mart:**

A data mart (DM) is a specialized version of a data warehouse (DW). Like data warehouses, data marts contain a snapshot of operational data that helps business people to strategize based on analyses of past trends and experiences. The key difference is that the creation of a data mart is predicated on a specific, predefined need for a certain grouping and configuration of select data. A data mart configuration emphasizes easy access to relevant information (Reference: Wiki). Data Marts are designed to help the manager make strategic decisions about their business.

**Writing Queries**

* DML, DDL, DCL
* INSERT, UPDATE, DELETE
* TRUNCATE
* Query settings
  + SET ANSI NULL ON - Specifies the null values for sql 92 standards.it determines the null value for true or false statements.
  + SET ARITHABORT - It stops executing the query while there is an overflow or divide by zero error.
  + SET ROW COUNT - Set row count stops processing the query after the specified no of rows. You can limit update up to a specified no.
  + SET ANSI padding - It controls the way columns stores the value which are shorter than the defined size of the columns.
  + SET ANSI WARNING - When on it displays the warning if there Is any null value for operations such as group by, order by max etc. and also if there is an arithmetic overflow. When off its displays no warning and also truncates the data to fit in the size of the column.
  + SET NO EXEC - Compiles query but doesn’t executes it
* SELECT

SELECT select\_list

[INTO new\_table\_]

FROM table\_source

[WHERE search\_condition]

[GROUP BY group\_by\_expression]

[HAVING search\_condition]

[ORDER BY order\_expression [ASC | DESC] ]

* Derived Table - having a select in your from clause that acts like a table, only around for the life of the query.
* Sub-query - usually in the where clause to present a shortened list of criteria to use for filtering
* WHERE vs. HAVING - The difference is, WHERE clause can only be applied on a static non-aggregated column whereas we will need to use HAVING for aggregated columns
* CASE -
  + Updates with Case

UPDATE [order details]

SET quantity =

(CASE

WHEN (quantity = 5) THEN 100

WHEN (quantity = 9) THEN 11

ELSE (quantity)

END)

* Joins are used in queries to explain how different tables are related. Joins also let you select data from a table depending upon data from another table.
  + Types of joins: INNER JOINs, OUTER JOINs, CROSS JOINs, SELF JOINS. OUTER JOINs are further classified as LEFT OUTER JOINS, RIGHT OUTER JOINS and FULL OUTER JOINS
  + INNER = match, OUTER = no match
  + self JOIN - can be used to do things like determine who a manager is in a single table of employees that includes empid and mgrid, recursion.
  + CROSS APPLY - returns only those rows in the outer table for which the table value function returns data
* Temp Tables
  + Types - local temporary tables (starting with #), global temporary tables (starting with ##), persistent temporary tables (prefixed by TempDB..), and table variables.(starting with (@)
* View - Physical Object, can be used to present denormalized data
* CTE common table expression - A temporary named result set (logical object) that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement

[WITH <common\_table\_expression> [,...]]

<common\_table\_expression>::=

cte\_name [(column\_name [,...])]

AS (cte\_query)

* Logical Operators- AND, OR, EXISTS, IN, LIKE, BETWEEN, NOT
* Testing for NULL: NULL IS NULL (True) *not* NULL = NULL (False), NULL <> NULL (False), instead use IS NOT NULL
* Functions
  + Aggregate - AVG, SUM, COUNT, MIN, MAX
  + Conversion - CAST/CONVERT
  + String - RTRIM, LTRIM, SUBSTRING, UPPER, LOWER, LEN, REPLACE, DIFFERENCE
* RANK - Ranking functions return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows. Ranking functions are nondeterministic. Ex shows 2nd highest salary by dept

select EmployeeID, EmployeeName, Department, Salary

from (

select

RANK() over (partition by Department order by Salary desc) as "Rank", \*

from @Employees

) as BaseTable

where Rank = 2

* WHILE LOOP

DECLARE @cnt INT = 0;  
WHILE @cnt < *cnt\_total*  
BEGIN  
 *{...statements...}*  
 SET @cnt = @cnt + 1;  
END;

* Triggers - Is like a special form of stored procedure that belongs to a table and is only *fired* when an INSERT, UPDATE or DELETE occurs
* Stored Procedures - A compiled database object that contains one or more T-SQL statements
  + SET NOCOUNT ON - Used to check whether the sql statement executed successfully or not. When no count is on then this message will not return and while it off then sql server returns the message.
* Functions - Functions accept parameters, perform some sort of action, and return a result
  + Scalar - Scalar functions return a single value; GETDATE, SUBSTRING
  + Table-Valued Functions (TVF) - Return a table
* Perf Tips
  + Avoid multiple joins
  + No cursors!
  + Create and use indexes
  + Use CASE instead of UPDATE
  + Only select the columns you need (avoid \*)
  + Only go to the source once
    - Use CTE, Temp Tables, Inline views
  + Delete and update in small batches
  + Pre-stage data - build dedicated tables for things like reporting where you don’t need up to the second data.
  + Avoid looping to use scalar functions
  + since non-sargable queries don’t scale, we need to find a better way to search for text. That’s where SQL Server’s Full Text Search comes in. Unfortunately, your queries must change – normally when you add an index, your queries just magically become faster. Full text indexes don’t work that way. You have to tweak your queries to use operators like CONTAINS instead of LIKE
    - sargable - Search Argument; Able to make use of a search argument to make use of index(es)
  + To check if something exists use EXISTS(SELECT…) or SELECT TOP 1...

**Question: Write a query to find the nth minimum and maximum**

**For Minimum**

Select \* From table t1 Where (n-1) = (Select Count(Distinct(t2.column)) From table t2 Where t2.column < t1.column)

**For maximum**

Select \* From table t1 Where (n-1) = (Select Count(Distinct(t2.column)) From table t2 Where t2.column > t1.column)

**Write a query to get the last record of the table.**

Select top 1 \* from table e1 order by id desc

**FizzBuzz - Count 1-100 for every # divisible by 3 print ‘FIZZ’ for every # div by 5 print ‘BUZZ’ for every number div by 3 & 5 print ‘FIZZBUZZ’**

DECLARE @NUMBER int;

SET @NUMBER = 1;

WHILE (@NUMBER <= 100)

BEGIN

IF

@NUMBER % 3 = 0 AND @NUMBER % 5 = 0

BEGIN

PRINT 'FIZZBUZZ'

END

ELSE

IF

@NUMBER % 3 = 0

BEGIN

PRINT 'FIZZ'

END

ELSE

IF

@NUMBER % 5 = 0

BEGIN

PRINT 'BUZZ'

END

ELSE

IF

@NUMBER % 3 <> 0 AND @NUMBER % 5 <> 0

BEGIN

PRINT @NUMBER

END

SET @NUMBER = @NUMBER + 1

END

**We can delete duplicate rows using CTE and ROW\_NUMBER () feature of SQL Server 2005 and SQL Server 2008.**

WITH CTE (COl1,Col2, DuplicateCount)

AS

(

SELECT COl1,Col2,

ROW\_NUMBER() OVER(PARTITION BY COl1,Col2 ORDER BY Col1) ASDuplicateCount

FROM DuplicateRcordTable

)

DELETE

FROM CTE

WHERE DuplicateCount >1

**Get Third Highest amount - Derived table**

SELECT \*

FROM

(SELECT \*,

RANK() OVER(ORDER BY Amount DESC) AS GrantRank

FROM [Grant] ) AS dt

WHERE GrantRank = 3

**OR**

WITH GI (seqno, RN) AS

(

SELECT seqno, ROW\_NUMBER() OVER (ORDER BY SeqNo) FROM GapsIslands

),

GI2 (MaxRN) AS

(

SELECT MAX(RN) as MaxRN FROM GI

)

--SELECT MaxRN FROM GI2

SELECT \* FROM GI

WHERE GI.RN = (SELECT MaxRN FROM GI2) - 2

**Find the Gaps**

WITH C AS

(

SELECT ID, SeqNo, ROW\_NUMBER() OVER(PARTITION BY ID ORDER BY SeqNo) AS rownum

FROM dbo.GapsIslands

)

SELECT Cur.ID, StartSeqNo=Cur.SeqNo + 1, EndSeqNo=Nxt.SeqNo - 1

FROM C AS Cur

JOIN C AS Nxt ON Cur.ID = Nxt.ID AND Nxt.rownum = Cur.rownum + 1

WHERE Nxt.SeqNo - Cur.SeqNo > 1

**Find the Islands**

SELECT StartSeqNo=MIN(SeqNo), EndSeqNo=MAX(SeqNo)

FROM (

SELECT SeqNo

,rn=SeqNo-ROW\_NUMBER() OVER (ORDER BY SeqNo)

FROM dbo.GapsIslands) a

GROUP BY rn;