**INFO 498**

**Summer 2016**

**Lecture Notes**

**July 26**

DATABASE DESIGN

Systematic process to design ultimately allows us to learn from our mistakes. We have phases and it tells us where we should be and when a phase is complete and we should move to the next phase.

Agile design is for software since the industry is so volatile. Create an iteration and release it every two weeks to make sure that what we are creating is still relevant.

Waterfall design each phase must be completely done before next phase is begun. Original method of design, still used for industries like construction. Used originally for software, was responsible for huge problems due to the speed of change in technology.

Prefer to only use INSERT states rather than UPDATE, slows down the system greatly when you have an absolute UPDATE as everything has to stop until that is completed. Need to understand how to store the data on disk so that INSERTs can be used to run a query and produce a report.

IMPORTING DATA

Can go to webpages and scrape data by doing simple copy and paste into Excel then formatting data into a useable format.

Right click on database, Tasks, then choose Import Data.

Edit Mapping allows you to change table values, data type, name of columns, etc.

During testing and validating of a new database you can scrape data that is relevant to the customer you are creating the database for in order to test it. Then you can truncate the data and send the empty database schema to the customer.

CONTROL OF FLOW

IF…ELSE – Imposes conditions on the execution of a transaction-SQL statement, condition is boolean. True statement follows an IF when executed, False statement follows an ELSE when executed.

Can combine IF…ELSE and BEGIN…END into statements.

WHILE loop – Issue a statement to run a code loop until a certain parameter occurs, such as run loop as long as value X is less than 5. Once it reaches 5 the loop is broken. You can control the loop from inside the loop with the BREAK and CONTINUE keywords. BREAK only breaks current loop, can have multiple loops coded within each other.

WAITFOR – Time element to specify how long to wait before executing code. Example, after alert is issued waiting at least 30 seconds before issuing another alert. Keywords – TIME, DELAY.

TRY….CATCH – Error reporting. Put group of statements in TRY block. If there is an error we CATCH all the errors from the block.

**July 27**

PROJECT STATUS

Create database by August 5th. Tables created, lookup tables populated. Stored procedures we created added to database, should have 4 per student by end of quarter. Create synthetic transactions.

When populating tables scrape some data from the internet.

Synthetic transactions due August 8th. Transactional table is where activity occurs, examples would be PRODUCT and ORDER tables from AW. Look at stored proc inside CUSTOMER\_BUILD to see code to generate random customers. Transaction should have:

* Have a parameter to reflect the number of rows to insert into a particular transactional table
* Calls a RAND() function to randomly generate PK values used to capture data in supporting tables
* Uses CASE statement to account for ‘random PK anomalies’ such as values of 0, 00, 000
* Uses a WHILE loop to process the number of rows passed-in via the parameter

CONTROL OF FLOW

GOTO – Alters the flow of execution to specified location. Statements that follow GOTO are skipped and processing continues at specified location. Can be used anywhere within a procedure, and can be nested.

ERROR-HANDLING

Goal is to anticipate failures. ‘Fail-early’ is better than ‘fail-late’. Need to manage communication to client. Potentially log the event for auditing.

SQL SERVER MESSAGES

* Message Number – Each error message has a number, most of the them are sysmessages in the master database. Message numbers 50001 and up are user-defined, lower are system defined. This is a hard-coded message, considered old-school method. RAISERROR.
* Severity Level – Number from 0-25. 0-10 is informational or a warning (not an error). 11-16 this is an error resulting from programming errors in SQL code. 17-25 indicates a resource problem, hardware problem, or internal problems in SQL server. 20 or higher you can terminate the connection entirely. Only administrator can use severity levels 17 and up.
* State – Internal for SQL server, not really used. Enter in 1 if using RAISERROR to pull up a message.
* Procedure – Object name in which the error occurred, could be stored procedure, check constraint, etc.
* Line – Code line where the error occurred, line of 0 indicates the problem occurred when the procedure was invoked.
* Message Text – Is the actual text of the error message, specifies what went wrong. Sometimes we don’t want to provide too much detail, ‘Please see your administrator’.

For error messages, if the message returned is the same you can use GOTO and only have to code the message once.

NESTED TRANSACTIONS

* One transaction is wrapped around another
  + Second transaction starts before first completes
  + Additional BEGIN TRAN statements
* @@trancount – variable that keeps track of which transactions in the statement we have open
  + Reflects the level of nested transactions
  + Each BEGIN TRAN increases count by 1
  + Each COMMIT TRAN decreases count by 1
  + Nothing is committed until count is 0
  + Error is raised by system if @@trancount is not what expected
  + Can be queried to determine level of nesting

**July 28**

Scaleability is the ability to do more. Want more people to be able to be processed, more sessions, more connections, more transactions. We add nodes, easier to process data with more ‘check-out lines’.

Availability is going to make multiple copies to prevent data loss in case a server is destroyed or corrupted, synchronous data transfer where we wait for confirmation from the second server location. Used to keep two servers in sync.

Data is moved to make more copies available to read to improve performance and to make more copies to protect ourselves from data loss. Often done together. Can also be moved for regional purposes in order to optimize global operations.

Data is also moved during development of changes to production. We move data back from the production environment and move newly developed schema forward. Environments are Dev, Test, Pre-Prod, and PROD. When moving data back from PROD you need to be careful because the data from PROD contains real world information that needs to be scrubbed so it is no longer sensitive. Different permissions in each environment, Devs vs. Ops.

Many types of tools for moving data, a dozen or so different methods.

* Copy of database (entire or portion)
  + Read purposes
  + Fault tolerance (safety against data-loss/downtime)
* Import large amounts of data
  + Marketing list
  + Product info from suppliers
* Export large amounts of data
  + Product info to customers
  + Send summary to data warehouse
* Movement of database/data files
  + Maintenance
    - Trace files
    - Moving backups to secure locations/lower environments
  + Expansion of environment
* If report would lock too many tables in an environment you can provide a copy of the database that the person running the report would be able to lock as much and as long as they want without impacting the business.

**August 1**

Output Parameter

Example:

CREATE PROCEDURE USPFindCust

@Fname varchar(35),

@Lname varchar(35),

@DOB DATE,

@CashSpent Numeric(7,2) OUTPUT,

@CustID INT OUTPUT

--Passing in data with input parameters, name & DOB, and getting different data back, dollars spent & CustID.

AS

SET @CashSpent = (SELECT SUM(PriceExtd)

FROM LINE\_ITEM LI

JOIN ORDER O ON LI.OrderID = O.OrderID

JOIN CUSTOMER C ON O.CustID = C.CustID

WHERE

DECLARE @Customer INT

DECLARE @CASH Numeric(7,2)

EXEC USPFindCust

@Fname = ‘Freda’, @Lname = ‘Walker’, @DOB = ‘3/16/1987’

@CustID = @Customer OUTPUT

@CashSpent = @CASH OUTPUT

Database Maintenance

Routine work to keep system running. Maintenance is part of the iterative process of database design.

Database Environment - 6 areas from INFO 340.

Defining System Vulnerabilities - What needs to be done to keep running?

* Define the events that will impact the database environment in the absence of simple maintenance
  + Hard drive fills up
  + Job dependencies
* The better DBA staff can understand the database environment, the better we are able to respond and take action.

Daily Tasks

* Daily backups of critical databases - Schedule this to reduce effort.
* Checking error logs
* Charting performance trends baselines

Maintenance Tasks

* Reduce vulnerabilities throughout the environment
* Some occur every 15 minutes like backups of the transaction log
* Some are once a day like a stand up meeting of entire DBA team or writing shift hand-off report

Goal is to build a routine set of tasks that improve the value of the database environment

* Cover all components
  + Hardware
  + Process
  + People

Key characteristics of Maintenance

* All maintenance is defined, scheduled and practiced
* Build a routine that is measureable (effectiveness)
* Assessed for improvements and flaws
* Optimized over time

General Topics

* Security - Review permissions to database objects
* Disaster Recovery
* System Performance
* Staff Skills

Maintenance Plan

SQL AGENT - Automated Wizard has common tasks like backup databases, shrink files, or rebuild indexes.

Alerts can be used when something goes wrong or when something is done correctly.

**August 2**

Mirroring - High-availability solution, copied behind the scenes to make an identical copy of the database. Synchronous transfer. 2-phase commit, needs to be confirmed prior to it being written. 1000x slower than non-synchronous transfer.

Replication - Partitioning data allows you to move region specific data to a server closer to users who commonly use that data.

Monitoring Databases

* SQL Server Profiler - Camera that shows what is going on, shows everything which can be overwhelming. Great to use to drill down into items. What is running, can do or schedule a trace to capture what is going on during a period of time. Can write trace data into the database or into a file, file uses less system resources. Not necessarily the first tool we use when troubleshooting.
* System monitor “PerfMon” - Windows OS monitoring tool, gives us hardware resources. How what is running is impacting the system. Can be represented as graph or report, can add variety of counters to track multiple items.
* Activity Monitor - What and/or who is connected to the database right now. If we run out of memory it substitutes disk space for memory, called paging. Good first tool to look at when initially troubleshooting. Dynamic, can filter data.
* Task Manager - High level system perspective, provides hardware cpu or ram specifics and information.
* DMVs - Gives insight into system data that is persistent. Can filter results, schedule. Specific views that are created with any new database, listed under Views/System Views.
* DMFs - Function that we put a value into in order to receive data from.

Monitoring system should be done when there is a problem and when there isn’t a problem, that allows you to create a benchmark. Slides from today show what DMVs are most commonly used/run, Perfmon counters to monitor.

GROUP BY

ROLLUP, CUBE, and GROUPING SETS

* Each can generate same result set as UNION ALL
* Usually more efficient than UNION ALL

ROLLUP

* Provides aggregate value for each level of hierarchy
* Under GROUP BY enter WITH ROLLUP
* Good when using dashboards, detailed reports

CUBE

* Calculates the results in every combination of columns
* Goes beyond structure of hierarchy (even more redundancy)
* Cartesian product

GROUPING SETS

* Can reduce ‘noise’ by specifying only the groupings desired
* Can have multiple grouping sets in a single query

**August 3**

Would want to move data if you are scaling the application by filtering data to regions that primarily access that data.

Create copies in case of server crashes or power outages.

ALERTS

Lets us know that something is occurring that we want to respond to.

SCALABILITY

Throughput Increase

* Number of transactions being completed
* Number of connections

Two types of scaling

* Scale-up - More hardware/resources in a single node.
* Scale-out - More nodes. Largest databases are scaled out, greater flexibility and higher end throughput. There are regional needs of data and functional needs of data, scale-out can help with both.

REPLICATION

Replication is a MS tool inside SQL Server that allows for the distribution of transactional data. Can send portions or an entire database anywhere that follows the metaphor of publishing industry like magazines.

* Publishers - Source of the data (often primary database server)
* Publications - Primary database (receives new or original data). Can vary in frequency, could be sent after every transactions or every three days depending on business need.
* Articles - Synonymous with table in primary database. This is filtered data, can be specific columns or tables that are sent.
* Distributors - Second server that manages all data movement
* Subscribers - Servers that receive copies of original data
* Subscriptions - Sign-up to receive publication

Different types of publications

* Snapshot - Sends a copy of the entire database so all history is provided.
* Transactional - As changes happen they are sent out
* Peer-to-peer - All nodes are publishers and subscribers, all read and write and share. Complex and rarely used.

**August 4**

Guest Speaker Tuesday - Founder and CEO of Wimmer Solutions. Billed out to random companies. Bring resume copies, ask about part-time jobs.

Functional or geographic partitioning, can be a mix of both depending on the business space of the office.

Replication allows you to filter data to users based on their pre-set demands and happens automatically once set up.

MIRRORING

Need High Availability

* Minimal downtime
* No data loss. Data loss is catastrophic.
* Synchronous data transfer.
* Heartbeat between two data centers, checks every second.
* Can redirect without even tripping an alert due to how quickly it switches.
* Automatic fail-over
* Synchronous with witness
* High-Performance, asynchronous with no witness. Faster, potential for data loss.
* High-Protection, synchronous with no witness. No automatic fail-over, humans have to get involved and downtime starts. No data loss but potential for downtime.

**August 8 & 9**

TROUBLESHOOTING

Refined set of skills to excel at troubleshooting:

* Patience - With people and process
* Confidence
* Ability to deduce and infer
* Ability to think under pressure
* Ability to listen
* Ability to communicate ‘effectively’
* Ability to delegate & trust

Rules:

* Be prepared
  + Know environment
  + Be able to leverage multiple tools
  + Be aware of co-worker skills
* Ask questions
* Take notes
* Ask for help when appropriate
* No cowboys in production!

Troubleshooting Steps

1. Communicate
   1. Notify chain of possible performance issues
   2. Be factual but relatively cryptic/vague
   3. Give ETA for next update
      1. 10 minutes for first update
      2. Every 30 minutes thereafter until resolved
2. Validate Issue
   1. Until validated issue is only a rumor
   2. Validate health of database
      1. Blocked spids: Activity Monitor
      2. Physical resources (CPU, Memory, I/O): Taskmgr
      3. Check SQL error logs and server system logs
   3. Execute scripts to verify user experience
      1. Best if already written
   4. Delegate task to others to assist
3. Define scope of issue
   1. What application(s) are not performing well?
   2. What are the exact symptoms?
   3. When did this begin?
   4. What tables are impacted in the database?
   5. Which objects touch these tables?
   6. What jobs/queries are currently running?
   7. Any open transactions? DBCC opentran
4. Define recent changes to environment
   1. What maintenance tasks in previous 24 hours?
   2. Any schema changes or deployment
      1. Dropped indexes, file groups or partitions?
   3. Any DELETEs or significant data archiving?
   4. Any IMPORT of large amount of data?
      1. Run UPDATE STATISTICS against affected tables
5. Compare to historical baselines
   1. WHERE is the system behaving poorly?
   2. If not obvious, look at middle-tier or app layer
   3. All web-boxes performing well?
   4. Send through a synthetic transaction
6. Be prepared
7. Learn
   1. Document issue
      1. What were the symptoms?
      2. What resolution tactics were tried?
      3. What resolution tactics were successful?

Getting better at troubleshooting

* Know environment
* Know product/platform
* Read online industry notes/blogs
* Build personal documentation
* Practice!!
  + Tools
  + Disasters

OPTIMIZATION

* Definitions
  + Page
    - Standard block of storage for databases
    - Maximum length of any row, 8,192 bytes (called ‘8k’)
  + Extent
    - Smallest block allocated by operating system
    - 8 contiguous pages
  + CPU
    - Core
    - Socket
    - Cache
    - Clock-speed
    - Bus
  + Spindle
    - Arm mechanism that touches disk media for reading and writing data
    - Moves like a turn-table to reach specific location to read/write
    - More is better (almost always)
      * Smaller-sized disks preferred over larger capacity
      * More ‘arms’ able to interact with media & satisfy requests
  + Clustered Index - Like a dictionary, inside the index
    - Physical order of data on disk
    - One per table
  + Non-Clustered Index
    - Traditional ordering of topics outside of data
    - Contains pointer to root node of clustered index
    - Unlimited (dozens?) per table
  + Index Scan
    - Every row in an index is read (possibly from disk)
    - Often expensive in CPU as well as I/O
    - Increasingly worse as table grows
    - Indicates that the structure of query to index is off
  + Index Seek
    - Efficient search of values through index (usually cached)
    - Stays efficient as table grows
    - Occurs when query and structure align
    - Indicates high-selectivity (<5% of values returned)
  + Hash - Only way for system to compare values in the absence of indexes.
    - Occurs when finding matches between 2 tables
    - Each row in table A compared to every row in table B
    - Exceptionally expensive in CPU, RAM and I/O
    - Increasingly worse as table grows
    - Indicates missing index
  + Page Split
    - Occurs when a particular page has filled and a row needs to be inserted for sorted order
    - Expensive yet necessary (what is alternative?)
    - Becomes an issue if excessive
  + Fill-Factor
    - Default % new page is filled with data on creation
    - OLTP is usually ~70%
    - Static/Read-Only and OLAP tables are always 100%
  + Fragmentation - Over 20% should consider a rebuild
    - Result of heavy DML (INSERT, UPDATE, DELETE)
  + Re-Index
    - Standard maintenance of busy tables
    - Some organizations rebuild several time a day
  + Statistics
    - Information to help query optimizer choose path
    - Can get out of date, Update Statistics
  + Caching
    - Random Access Memory (RAM)
  + Buffer Cache
    - Working set of most-recently accessed data
    - All database sessions share this space
  + LRU Caching Algorithm
    - Most-recently used (‘MRU’ or ‘hot’)
    - Least-recently used (‘LRU’ or ‘cold’)
    - Common-method of memory management across computing (not just for databases)
    - While logical I/O is hundreds-times faster than physical I/O it cannot be considered ‘free’
      * Still need to be concerned with unnecessary logical I/O
  + Cache Hit
    - Finding desired data in memory
  + Self-caching
    - Rows are brought into memory by the same query
  + Page file
    - File on hard disk
    - Dedicated spot on disk for data dropped from RAM
    - Sort of like a storage unit as support for garage, still slow
  + Page Fault
    - Soft
      * Not in database memory address space
      * Data retrieved from somewhere else in memory
    - Hard
      * Not in memory at all
      * Data retrieved from disk
  + Want to clump the ‘hot’ data together
  + Tables are of four different types
    - Continuous Growth - Order table
      * High volume of INSERT activity with few DELETEs
      * Value/usefulness of data often wanes with age
        + When newest rows are ‘hottest’ good performance occurs if indexing strategy avoids older rows
        + Clustered index on sequential value (PK or time stamp)
      * A query that touches most rows before filter
        + Performance will degrade over time
        + Great (!!) candidate for archival of derelict rows
    - Purge Eldest - Archive old unused data
      * Rows are deleted fairly consistently
      * Oldest rows are first to get deleted
      * Maintains sequential order but has potential gap
      * Able to keep in cache relatively easily
    - Purge Not-by-Age - Archive based on status, ticket has been resolved
      * Rows are DELETED but by another criteria than age
      * Scatters data across pages with potential gaps
      * Difficult to keep in cache
    - Complete Purge/Re-grow - Class list, start from 0 each time there is a new class
      * Similar to continuous growth
      * Must maintain gaps with truncate/drop & recreate
      * Potential for huge performance drag if not watched
    - Maintenance, log backups, indexing are all changed based on the type of table
* Configuration
  + Recovery Model
    - Full
    - Simple
    - Bulk-Logged
  + TEMPDB
    - Equal number of .mdf/.ndf as count of CPU cores, every core can operate a different data file
  + Auto-grow
    - 64k blocks
    - Rarely use percentage
  + File Groups
    - Collection of .mdf files
    - Can be placed on separate partitions
    - Separate READ-centric tables from WRITE-centric
    - Separate busy from less busy
    - Separate sequential (log files) from random
    - Separate user databases from system databases
* Data Access Basics
  + Caching
* Indexes - Clustered vs Non-clustered
* Execution Plans

**August 10**

SCALABILITY

Replication - Uses filtering to allow for scalability and improved efficiency.

* Geographic partitioning
* Functional partitioning

Log Shipping - No filtering.

* Primary and secondary databases, take full backup of primary and copy to secondary
* Then every 15 minutes take a backup of the transaction log and restore it to secondary
* In old days this was called a ‘warm backup’, no longer acceptable to have data loss
* Secondary database is READ-ONLY, by routing reads to this database and writing to primary database you greatly improve efficiency by eliminating lock contention between the different tasks
* Reads have shared locks, writes are exclusive locks.
  + Intent exclusive locks while waiting in line for transaction to be processed
  + Negotiate who gets to go first
  + Lock escalations are when you need to lock more than originally anticipated. If you get a lot of these then the statistics your system is using for its original estimate are off
  + Reads benefit from indexes, writes do not. Want to have few indexes on the primary database

DISASTER RECOVERY - Can take many forms, need to be able to restore when one happens

* Strategy
  + Analyze the environment
  + Consider risks
  + Build redundancy
  + Practice recovery
  + Document everything
  + Leads to Fault Tolerance
    - Hardware
      * Redundant power supplies, etc
    - Software
      * Anchoring servers to steel beams
    - Data
    - People
      * Create SOP and document so knowledge is not lost
* Types of disasters
  + Hacking - 80% of budget spent to prevent this, 2% of disasters are due to this
  + Natural Disaster
  + Component Failure
  + Human Error - 80% of all disasters are due to this, 2% of budget spent to prevent this
    - Need to focus and have more training, don’t rush anything

CLUSTERING

* Two servers, one active and one passive. Passive database does nothing but wait for active server to fail. All connections write to a VIP (virtual IP address). All transactions are written to a disk shared by both servers.
* For older hardware or highly sensitive information, like banking information.
* Takes 10 minutes to switch from active to passive database, all databases have to transfer over. Automatic. Mirroring only one database at a time switches; also don’t have shared hardware.
* Still vulnerable to a site disaster.
* For redundancy of servers. Mirroring is cheaper.

SNAPSHOT BACKUP

* Not a full backup, creates a restore point in the production system
* Assumes that the fundamental database is not damaged
* Intended to be used during releases to avoid the time it would have taken to create a full backup of the database
* Create an instantaneous empty sparse file that anything that gets destroyed goes into

**August 11**

INDEXES

Clustered - Physical sorting of data. 1 per table. Able to remove the clustered index, then table is called a ‘heap’. You want to kill the largest number of rows as possible as early as possible, larger tables first.

* Clustered index should be on a narrow column
* Sequential is good, automatically placed on PK column, timestamp is also a good option.
* Never changes
* Having a clustered index in a WRITE centric environment allows us to write very quickly
* Index Structure
  + Entry point is called a Root Node or Clustering Key
  + First level down is called the Intermediate Node
  + Second level down are called Leaf Nodes, data is stored at the leaf nodes.
  + At third level of non-clustered index in order to find the clustering key
* In a READ centric environment it depends on what your query is. If you are seeking a range of information then a clustered index works better, for specific data then a non-clustered might be better.
* Data can be pulled at the extent level in blocks during a clustered index search.

RAID - Redundant Array Independent Disks

We are worried about performance, efficiency, and stability of the system

* Redundant leads to speed and stability
* Writing in parallel

RAID Levels

* 0 - Striping, fast, parallel writes and reads, does not lead to stability because if one drives fails the whole set is dead
* 1 - Mirroring, increases stability but expensive as you have to double the hardware, known as a 0+1 because it is a striped mirror. Parallel parallel reads since we can read from primary and mirrored server. Also known as n/2, total capacity divided by 2
* 5 - Doesn’t mirror, add a Parity drive. We first do striping to all other drives, then at end write to parity drive. Known as n - 1. Can sustain 1 drive failure, but not 2. Only half as fast on writes since you have to write twice. Striping with parity.
* A collection of physical disks lumped together in LUNs, want to separate WRITE from READ drives.

VIEWS

Very common when reports are being generated.

* A table
* Result of a query, “SELECT”
* Lives only in memory
* Simplifies environment - Allows us to change the names of objects, increases security as user can be given access to a view without allowing access to the database
* Intended for less sophisticated people
* Can be used for applications, however, view has to be recreated whenever it is run and this affects performance if repeated frequently

**August 17**

Study Guide Answers

**1.** We evaluate this because it is going to affect how we index and archive. Volatile data would get its own physical location on disk. Maintenance concerns - Data access and where we place indexes.

Continuous Growth - Order table, archive by quarter and put each old quarter in its own physical location.

Purge Eldest - Want to have the ‘hot’ tables in memory, index by date. If we purge by eldest there will be gaps in data. Need to deal with fragmentation issues.

Purge Other Than Eldest - Ticketing system, archive data once ticket has been completed. Gaps will be created when rows are deleted after order has been completed. Could change physical ordering by something like status.

Truncate & Rebuild - Goal is to keep static data in memory, ex is current price list. Once it goes away a new updated price list will take its place. Index by name, update statistics immediately once rebuilt.

**2.** Insert anomaly - May have two nouns in one table, student/dorm in STUDENT. Cannot include dorm information until a student is registered for that dorm.

Delete anomaly - If we get rid of the student we lose all dorm information, people put in fake people in order to hold the dorm information.

Update anomaly - Cannot make changes without both a student and dorm existing.

With FK we can reference all the dorm information with just the dorm ID without bringing all the dorm information into the Student table.

**3.**  Differential backup is the change in data that has been created since the last full backup. Transaction log backup is data since the last backup of any kind. Weekly full backup, nightly differential, every 15 minutes a transaction log. Then take a manual backup of the transaction log if available to bring database up to current point in time.

**4.**

* Communicate - Report that you are investigating and ask for reports of issue.
* Validate - Verify that there is a problem with the database, communicate that if you find evidence.
* Use monitoring tools to try and determine what the issue is. Don’t want to solve a symptom, try to find the source of the problem and resolve it. Iterative process.

**5.** Want to separate write and read activity. OLTP is Write-centric, live data, more tables, few indexes, many users, highly available, so when tables are locked during a write there are more tables still open for use. OLAP is Read-centric, de-normalized, large number of indexes, dead data, few users.

**6.** Transactions - How does data come in, volume of data, peak hours

* Customers - Internal and external, know what their needs and concerns are.
* Objects - Know key tables and procedures, functions, constraints.
* DBA skills - Know strengths and weaknesses of staff.
* Hardware - Know where servers are stored, data warehouses are.
* Data flow - Look up

**7.** Know your environment and evaluate risks to environment. Try to eliminate single points of failure, document and then practice disaster recovery to test its effectiveness and gain experience.

**8.** DMVs - sysdm.exexsession, osperformancecounters, indexphysicalstats, look at views under SQL and starts with dm.

**9.** Fault Tolerance - Ability to take a failure and continue processing, database mirroring - synchronous data transfer, clustering is fault tolerance for server failure, RAID is fault tolerance for disk failures, dual raid cards to avoid hardware failure.

**10**. Clustered - Sorted order, only 1 per table, ranged search, data at leaf level.

Non-clustered - Best for unique values, pointer at leaf level, unlimited number per table, works well with FKs.

**11.** Write-ahead logging, write to transaction log first to ensure ACID principle of durability. Checkpoint - Take whatever has been written to the transaction log every couple of minutes and cement it to disk. Synchronous data transfer from data mirroring. Begin Tran/Commit Tran, THROW or RAISERROR.

**12.** Begin/End

IF/ELSE

BREAK/CONTINUE

WHILE

WAITFOR

CASE

TRY/CATCH

GOTO

**13.** Database mirroring - High availability, automatic failover, guaranteed no data-loss, minimal downtime.

Log Shipping - Scalability solution - Backup copy and load of entire database with no filtering, can have security issues since it is the entire database. Read-only, good to use to separate read and write activity.

Replication - Scalability solution - Filtering, partition by column, row, functional, geographic. Can be very complex.

**14.** LRU - Least recently used algorithm, when memory is full the oldest accessed object is sent to a specific area on disk in order to create space. If the number assigned to objects in memory is wrong then it is a ‘soft page fault’ and memory has to be scanned. Hard page fault is when an expected object in memory is not there at all.

**15.** Understand the environment, interviewed customers, know data consumption, evaluated vulnerabilities, take steps to be proactive with maintenance to reduce risk. Planned out, documented, scheduled/practiced, evaluate performance and efficiency of maintenance, optimize over time.

**16.** Star schema - De-normalized, central fact table surrounded by dimensions. Answer questions around each fact. Who, What, Where, How.

Snowflake - Normalized

Star Flake - Hybrid table of star/snowflake.

Fact table - 80% of data will be stored in fact table, composite primary key.

**17.** Visual demonstration of how the query engine is going to disk to retrieve data. Tell us if we are scan/seek, which indexes were used, what percent of query was used in each step, number of rows filtered out, any hashing, any joined tables. Allows us to understand which component in the query was most expensive and try to optimize it. Bottom right to upper left.

**18.** Logical read is in memory, physical read is on disk.

**19.** Index seek is seeking out specific rows in a table, index scan we have to scan every row for every data.

**20.** RAID 0 - Parallel writes, striping, super fast, vulnerable as one failure breaks the entire RAID.

RAID 1 - Mirrored disk, parallel reads and writes, fast but not as fast.

RAID 5 - Striping plus parity, one disk where data is striped across drives and one drives everything is written. Can absorb one failure. Twice as expensive when writing, bad for OLTP.

RAID 0 + 1, mirrored stripe.

**21.** Data warehouse - Warehouse for entire organization, enterprise warehouse.

Data mart - Warehouse for a single department, faster start-up, quicker ROI, smaller in scope.

**22.** Synchronous - Two phase commit, wait for signal that copy has been received before committing a transaction. Asynchronous introduces a lag between primary server and backup, much faster but risk of data loss.

**23.** Profiler - Satellite view, lets us see everything going on in the database. Don’t start troubleshooting with this option.

PerfMon

DMV

Task Manager

Activity Monitor

**24**. Checkpoint - Every couple of minutes transaction log is transferred from memory to disk, cemented.

Fill factor - Percent of page that will be filled when page is created.

Page split - When a row needs to be added to a page that is 100% full, divide page into two new pages.

Page fault - When we look for an object in memory, if in memory but at bad address is ‘soft’ fault, if item is not in memory at all then it is ‘hard’ fault.

**Black Coffee Questions Structure**

1. Staff, not Person, pass in position name (exists already) first name/last name/email/phone