**QUESTIONS for DON**

When would you use:

**NULL (rules)**

* Always use ISNULL or ISNOTNULL to accommodate for numeric and string nulls
  + Used mostly on numeric calculations
* When aggregating(summing) NULLS are excluded
* Count(\*) nulls included
* Count(FieldName) nulls excluded
* Null comparison will always fail (false)
  + Use isnull: where isnull(atr.TierID, 0) !=1
* When grouping, Nulls will be grouped together
* When averaging NULLS are excluded unless using ISNULL

select avg(v.Value), sum(v.Value), avg(isnull(v.Value, 0))

* + from ( values (2), (null), (4)) as v(Value)
  + returns 3, 6, 2

**NULLIF (rules)**

**This first changes an empty string into a NULL. Then it changes all NULL’s into the string “Unknown”. This is a trick to accommodate both empty strings and NULL’s.**

Select isnull(nullif(cl.ReportingParty, ''), 'Unknown') as ReportedBy,

**Dups**

* How would you accommodate for duplicates?
  + The best way is you need to determine which fields constitute a unique record. This will return no dups.
  + Use the windows function to row number the dups then filter for 1 only in where clause.
    - Make sure you are partitioning and ordering on the correct fields or your results will be incorrect.
  + The keyword distinct is another way to accommodate for dups.
  + Dump the data into excel to check for dups.
* Subquery in select
  + If I just need to see if something exists in another table then use exists but if I need more then that use either a derived table in the FROM/Joins or a CTE or a temp table.
    - EXAMPLE – If I want to see if a claim was ever in litigation then use WHERE EXISTS and select from the History table but if I need to know more such as a date range then use a derived table or a CTE or a temp table.
  + A derived table in the FROM/joins is the same as a CTE but performs differently
* Subquery in a join
  + It is better to use WHERE EXISTS then to use a sub-query.
* Create #Table Insert values into #Table vs Create #Temp derived table
  + If I need to INDEX for speed use Create #Table Insert Values

create table #Queue (

QueueID integer,

TypeID integer,

StatusID integer,

FollowUpDate date,

FollowUpReminder bit,

CreatedBy integer,

CreatedDate datetime,

[Priority] integer,

AssignedTo integer,

AssignedToRoleID integer,

Age integer)

create clustered index idx\_QueueID on #Queue (QueueID)

create nonclustered index idx\_TypeID on #Queue (TypeID)

* + - INDEX does not work on derived tables or CTE
  + Intellisense does not work on #Temp tables
* Why use a CTE verses #TEMP table and unions
  + Use a CTE when:
    - Don’t need indexes
    - Result set is not large
  + Use a #TEMP table when:
    - You need to use an index
    - Result set is large
    - CTE code is large and needs to be broken down
* Table variable
  + Can use an index
* Subquery/sub-select verses joins
* Why do you join on a sub query
* Full OUTER join
* Recursion
  + Use for Hierarchical data
* Cross join
  + Use if I have nothing to join on

**How to design SP**

* When to use union append verses joining CTEs
  + Use Unions when you want both detail and summary in the SP
    - You can summarize/aggregate in the SP or SSRS/DOMO
  + You almost always use CTEs and join them
* When to use #Temp tables only, CTEs only or both
* When to use derived tables
* When to use multiple CTEs
* When to use multiple CTEs then end the multiple CTEs
* When to use Cross Join
* When to use full outer join

**Types of reports**

* YTD report
  + Uses @FirstOfYear and @EndingDate
  + This can be hard coded
* Date Range report
  + Uses @StartingDate and @EndingDate
* Point In Time report
  + Uses @AsofDate parameter
* YOY report
  + Uses Date Range
  + Don uses Recursive Date Range CTE

**Types of Tables used in SPs**

* Create #Table insert values
  + **Can** Index for speed
* Select Into #Temp Table
  + **Can** Index for speed
* Derived table in Join
  + **Cannot** Index for speed
* Table Variable
  + **Can** Index for speed

**Dates**

* When Grouping/Aggregating on dates always cast to Date
  + Aggregating on DateTime is useless