**Universal TSQL Code to learn**

**This code tells us if a claim has ever in its lifetime been in Litigation Status. This uses the ClaimStatusHistory table. This covers both current and History status.**

  select ClaimID,

         ClaimNumber,

         Litigation = iif(c.ClaimStatusID = 3 or exists( select \* from dbo.ClaimStatusHistory as csh where csh.ClaimStatusID = 3 and csh.ClaimID = c.ClaimID ), 'yes', 'no')

from          dbo.Claim as c

**Formatting as %**

format(isnull(cast(sm.[Day\_30] as decimal(20,5)) / nullif(bs.PoliciesSold, 0), 0), 'P2') as [Day\_30],

format(isnull(cast(sm.[Day\_60] as decimal(20,5)) / nullif(bs.PoliciesSold, 0), 0), 'P2') as [Day\_60],

format(isnull(cast(sm.[Day\_90] as decimal(20,5)) / nullif(bs.PoliciesSold, 0), 0), 'P2') as [Day\_90],

format(isnull(cast(sm.[Day\_120] as decimal(20,5)) / nullif(bs.PoliciesSold, 0), 0), 'P2') as [Day\_120]

**NullIf**

**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,

nullif((isnull(ibs.TotalClaimCount, 0) - isnull(cbs.ClosedClaimCount, 0)), 0)

ISNULL(cbs.TotalClosedClaimsIncurred, 0) / nullif(isnull(cbs.ClosedClaimCount,0), 0) AS ClosedClaimsAVG,

**Averaging with NULLS**

**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)

**CharIndex**

select

c.PolicyId

,Common.dbo.GroupConcat(case

when charindex('Agent Request', c.ReasonForCancellation) > 0 then 'Agent Request'

when charindex('Binder Cancellation', c.ReasonForCancellation) > 0 then 'Binder Cancellation'

when charindex('Incorrect statement on the application: Policy is VOID ab initio', c.ReasonForCancellation) > 0 then 'Ab initio'

when charindex('Lapse', c.ReasonForCancellation) > 0 then 'Lapse'

when charindex('Non Renewal', c.ReasonForCancellation) > 0 then 'Non Renewal'

when charindex('Non-Payment of Premium', c.ReasonForCancellation) > 0 then 'Non-Payment of Premium'

when charindex('Policy Holder Request', c.ReasonForCancellation) > 0 then 'Policy Holder Request'

when charindex('no application received', c.ReasonForCancellation) > 0 then 'No Application'

when c.ReasonForCancellation is null then null

else 'Other'

end, ', ') as CancelType

from Atlas.dbo.Cancellations as c

**Accommodate for DUPS**

--row\_number() over (partition by c.ClaimID, ie.ExaminerSystemUserID order by q.[QueueID]) as DuplicateCount,

**Cross Apply**

from ( select ie.ExaminerSystemUserID,

ie.ExaminerName,

ie.ExaminerTitle,

l.[Location],

v.ConcernGroup

from dbo.InternalExaminers as ie

cross apply ( values ('INTERIM'), ('FINAL'), ('SUPPLEMENT')) as v(ConcernGroup)

cross apply ( select distinct

[Location]

from dbo.InternalExaminers) as l([Location])) as ie

left join #ExaminerDetailAssignments as ed

on ed.ExaminerSystemUserID = ie.ExaminerSystemUserID

and ed.ConcernGroup = ie.ConcernGroup

and ed.[Location] = ie.[Location]

**This code uses a join to call the split function instead of the WHERE clause**

ALTER procedure [dbo].[reportCancelationsByReason] '20160101', '20170110', 'FL16905', '1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17' --'Policy Holder Request ,Failure to comply with underwriting requirements, a complete application was not received within the required time frame.,Binder cancellation for non payment of initial premium,Policy Holder Request,Agent Request,Renewal payment not received'

@StartDate date,

@EndDate date,

@AgencyCode varchar(100),

@CancelationReason varchar(max)

as

begin

set nocount on

if @StartDate > @EndDate return

select distinct

p.policynumber,

pd.Name as FormType,

dbo.GetPolicyInsureds(p.policyid) as Insured,

rtrim(ad.AddressLine1 + ' ' + ad.AddressLine2) as Address,

ad.City,

ad.ZipCode,

--REPLACE(REPLACE(isnull(c.ReasonForCancellation,''), CHAR(13), ''), CHAR(10), ' ') as ReasonForCancel,

isnull(c.ReasonForCancellation,'') as ReasonForCancel,

c.CancellationEffectiveDate

from cancellations c

join policy p on p.policyid = c.PolicyID

join agency a on a.Agencyid=p.AgencyID

join product pd on pd.ProductID = p.ProductID

join InsuredEntity ie on ie.PolicyID = p.PolicyID

join Addresses ad on ad.AddressId = ie.PropertyAddressID

inner join dbo.GetCancelationReasons() as cx

inner join dbo.Split(@CancelationReason, ',') as s

on s.StringValue = cx.typeID

on REPLACE(REPLACE(isnull(c.ReasonForCancellation,''), CHAR(13), ''), CHAR(10), ' ') like cx.ReasonForCancellation + '%'

where

a.AgencyCode = @AgencyCode

and c.CancellationEffectiveDate between @StartDate and @EndDate

--and REPLACE(REPLACE(isnull(c.ReasonForCancellation,''), CHAR(13), ''), CHAR(10), ' ') IN (SELECT StringValue FROM dbo.Split(@CancelationReason,','))

--and charindex(REPLACE(REPLACE(isnull(c.ReasonForCancellation,''), CHAR(13), ''), CHAR(10), ' '), @CancelationReason) > 0

and c.Active = 1

order by CancellationEffectiveDate

end

**This code replaces carriage return and line feed to empty string**

REPLACE(REPLACE(isnull(c.ReasonForCancellation,''), CHAR(13), ''), CHAR(10), ' ') as ReasonForCancel

**NULL**

When there are null values in a record they will fail when comparing to a number i.e. !=1. A comparison can be in the WHERE clause or the SELECT. As a general rule until I really know the data, Always accommodate for NULL

* When comparing numbers
* When comparing ID’s
* Be aware that some situations you want a NULL value
  + Need to research these situations.

So use ISNULL to accommodate for this.

where isnull(atr.TierID, 0) !=1 --and atr.TierID != 1

**Code snippets**

This should get you the adjuster:

select

concat(su.FirstName + ' ', su.LastName) as Adjuster

from            Claim as c

left join       dbo.SystemUsers as su

        on      su.SystemUserID = dbo.GetAdjusterByClaimID(c.ClaimID)

--THIS USES A SELF JOIN

Assignments

as (

select ors.ClaimID,

upper(isnull(adn.FirstName, ado.FirstName) + ' ' +

isnull(adn.LastName, ado.LastName)) as Adjuster,

upper(coalesce(ben.BusinessName, adn.BusinessName, beo.BusinessName, ado.BusinessName)) as AdjustingCompany

from OpenReserves as ors

inner join ClaimDetailIDs as cdi

inner join ClaimDetails as cd

left join dbo.SystemUserContactAssoc as sca

inner join dbo.Contact as adn

left join dbo.Contact as ben

on ben.ContactID = adn.ParentContactID

on adn.ContactID = sca.ContactID

on sca.SystemUserID = cd.AdjusterSystemUserID

inner join dbo.Contact as ado

left join dbo.Contact as beo

on beo.ContactID = ado.ParentContactID

on ado.ContactID = cd.AdjustingCompanyContactID

on cdi.ClaimDetailID = cd.ClaimDetailID

on cdi.ClaimID = ors.ClaimID

),

**--**

select co.CompanyID,

co.CompanyCode

from dbo.Company as co

where charindex(',' + cast(co.CompanyID as varchar(10)) + ',',

',' + isnull(@CompanyID, cast(co.CompanyID as varchar(10))) + ',') > 0

**--**

sum(dbo.UnearnedPremium(wp.Premium, @StartDate, wp.WPEffectiveDate, wp.WPExpirationDate)) as Unearned

**OVER PARTITION (means a grouping portioned by row numbers or rank ect.)**

row\_number() over (partition by t.CompanyID, c.CompanyCode, s.StateCode, co.CountyDescription, t.Territory order by left(pr.Name, 2) desc, right(pr.Name, 1)) as ProductID

select row\_number() over (order by count(p.PolicyId) desc, sum(pva.SubTotal) desc) as [Rank]

RowIndex = row\_number() over (partition by a.ClaimId order by a.Active desc -- prioritize active records ,a.CreatedDate -- prioritize oldest records) from assignments as a

rank() over (partition by r.RepID, isnull(ty.[Month], ly.[Month]) order by isnull(ty.Premium, 0) - isnull(ly.Premium, 0) desc) as Ranking,

**GROUP BY vs PARTITION BY**

select \* from callerlog

Output will be

Code:

CALLERID CALLER CALLEDTO CALLER\_DT OUTGOING\_SEC

3 456 123 02/21/2013 10

4 123 678 02/21/2013 20

1 123 456 02/21/2013 30

2 123 456 02/21/2013 10

5 678 456 02/21/2013 40

6 456 678 02/20/2013 20

7 789 555 02/21/2013 10

8 567 321 02/21/2013 100

We have used below in one of the article  
  
Query by using group by

Code:

select caller ,count(\*) as outgoing from callerlog group by caller

Output will be as below

Code:

CALLER OUTGOING

123 3

456 2

567 1

678 1

789 1

Group by has reduced the records from **8** to **5**.  
  
  
Query by using partition by

Code:

select caller ,count(\*) over (partition by caller ) as outgoing from callerlog

Output will be as below

Code:

CALLER OUTGOING

123 3

123 3

123 3

456 2

456 2

567 1

678 1

789 1

Partition by has not reduced the no. of records. Remains same.  
  
Below query we will not workout for group by

**PIVOT**

select

p.\*

from

(

select

d.Name

,isnull(sum(wp.Premium),0) as [WrittenPremium]

from @dates as d

left join Atlas.dbo.WrittenPremium as wp

join Atlas.dbo.AccountingTransaction as at

join Atlas.dbo.Policy as p

join Atlas.dbo.Agency as a

join @agencies as aa

on aa.agencyid = a.AgencyID

on a.AgencyID = p.AgencyID

on p.PolicyID = at.PolicyID

and (@companyId is null or p.CompanyID = @companyId)

on at.AccountingTransactionID = wp.AccountingTransactionID

on case when wp.BookedDate > wp.WPEffectiveDate then wp.BookedDate else wp.WPEffectiveDate end between d.StartDate and d.EndDate

and wp.FeeID = 0

group by d.Name

) as q

pivot

(

max([WrittenPremium]) for Name in

(

[YTD]

,[Last YTD]

,[Last Year]

)

) as p

**IIF**

iif(left(pr.[Name], 2) = 'HO', 'Homeowners', iif(left(pr.[Name], 2) = 'DP', 'Dwelling', 'Inland Marine')) as LineOfBusiness,

sum(iif(cd.Outstanding = 0 and cd.Losses != 0, 1, 0)) as ClosedWithPayment,

sum(iif(cd.Outstanding = 0 and cd.Losses = 0, 1, 0)) as ClosedWithoutPayment,

sum(iif(cd.Outstanding != 0, 1, 0)) as [Open],

ltrim(concat(format(nullif(tcd.TimeOff, 0), 'N2') + ' hour' + iif(tcd.TimeOff != 1, 's', '') + ' time off.',

iif(tcd.OvertimeHours >= tcd.RegularHours \* .50, ' High overtime.', ''))) as Comment

isnull(sum(iif(sm.ClaimsDraftPrintLocation = 'FieldDraft', sm.[Count], 0)), 0) as YTDClaimsWFieldDrafts,

isnull(sum(iif(sm.ClaimsDraftPrintLocation = 'FieldDraft', 0, sm.[Count])), 0) as YTDClaimsWInternalDrafts,

**AVG**

round(id.Premium / id.[Count], 2) as avgPremium,

round(id.TIV / id.[Count], 2) as avgTIV

round(sum(id.Premium) / sum(id.[Count]), 2) as avgPremium,

round(sum(id.TIV) / sum(id.[Count]), 2) as avgTIV

**MAXRECURSION number**  
Specifies the maximum number of recursions allowed for this query. number is a nonnegative integer between 0 and 32767. When 0 is specified, no limit is applied. If this option is not specified, the default limit for the server is 100.

When the specified or default number for MAXRECURSION limit is reached during query execution, the query is ended and an error is returned.

Because of this error, all effects of the statement are rolled back. If the statement is a SELECT statement, partial results or no results may be returned. Any partial results returned may not include all rows on recursion levels beyond the specified maximum recursion level.

MAXRECURSION can be used to prevent a poorly formed recursive common table expression from entering into an infinite loop

--Creates an infinite loop

WITH cte (CustomerID, PersonID, StoreID) AS

(

SELECT CustomerID, PersonID, StoreID

FROM Sales.Customer

WHERE PersonID IS NOT NULL

UNION ALL

SELECT cte.CustomerID, cte.PersonID, cte.StoreID

FROM cte

JOIN Sales.Customer AS e

ON cte.PersonID = e.CustomerID

)

--Uses MAXRECURSION to limit the recursive levels to 2

SELECT CustomerID, PersonID, StoreID

FROM cte

OPTION (MAXRECURSION 2);

GO

**RECURSIVE QUERY**

with agencies as

(

select

a.AgencyId

,a.Active

,aa.ParentId

from Atlas.dbo.Agency as a

left join Atlas.dbo.AgencyAssoc as aa

on aa.ChildId = a.AgencyID

and aa.RemovedDate is null

)

,agencyAncestryRecursive as

(

select

a.AgencyID

,a.AgencyID as AncestorAgencyId

,a.ParentId

,1 as AscendantId

from agencies as a

union all

select

aar.AgencyID

,a.AgencyId

,a.ParentId

,aar.AscendantId + 1

from agencyAncestryRecursive as aar

join agencies as a

on a.AgencyID = aar.ParentId

and a.Active = 1 -- ensure that any parent relationship refers to an active agency...

)