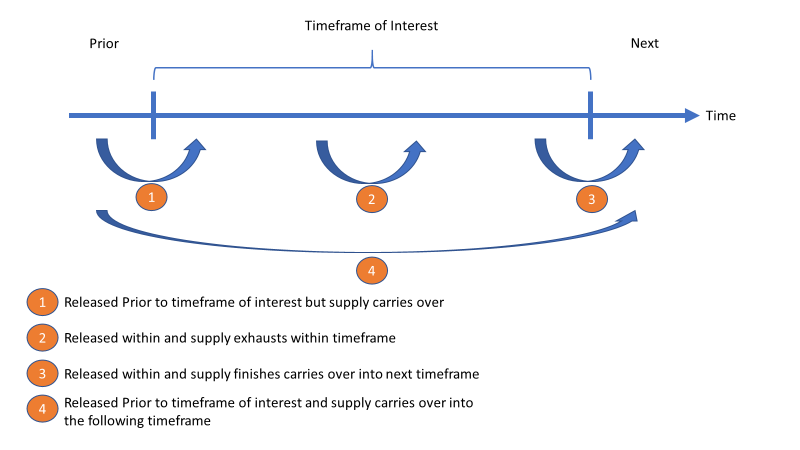
# Proportion Days Covered (PDC) & Concomitant Drug Utilization with SQL

Calculation of PDC can be efficiently performed within SQL by using dynamic SQL. But before that can be applied, the data must first be standardized to a timeframe of reference. We need to carve out specific business rules to identify our timeframe of interest and the methodology for calculating PDC.

1. Where is the starting point for the timeframe?
   1. Last “x” days?
   2. Last “x” calendar years?
   3. Beginning of each Fiscal Quarter?
2. Where is the ending point for the timeframe?
   1. Today?
   2. End of a calendar year?
   3. End of a Fiscal Quarter?
3. Where do we start counting PDC?
   1. The start of the timeframe?
      1. CON: Do we want to negatively score a patient who hasn’t received the medication yet?
   2. From when the patient first has supply on hand within the timeframe?
      1. PRO: Doesn’t count against the patient’s PDC until they have supply
      2. CON: Shorter timeframes might inflate the PDC
4. Where do we count until?
   1. The end of the timeframe?
      1. CON: Change / discontinuation of therapy might look like low PDC.
   2. The end of the latest prescription?
      1. CON: It’ll miss patients who need a refill and inflate PDC

Regardless of the specific timeframe you choose, the below graphic is the situation you will next encounter regarding the prescription fill data.

**Figure 1. All Possible Combinations of Prescription Fills within the Timeframe of Interest**



# Step 1 – Standardize the Raw Data

Using your start and stop definitions defined above, we can now create a function that calculates the Modified Days Supply as the Day Supply within the Timeframe of Interest and a Modified Fill Date as the First Day the patient had medication on hand within the Timeframe of Interest by addressing the 4 possible conditions of interest and ignoring all else.

For my purposes, I generally trend data on a quarterly basis so the function below is specific to Fiscal Year and Fiscal Quarter:

create function [FX].[RxQuarter\_Int] (@Date date, @DaysSupply int, @FYFQ int)

returns int

begin

declare @NewDate date = DateAdd(dd, @DaysSupply, @Date)

declare @MaxDate date = (select maxDate from Staging.QuarterDates where FYFQ = @FYFQ) declare @MinDate date = (select minDate from Staging.QuarterDates where FYFQ = @FYFQ) declare @ModifiedDaysSupply int

-- begins before the quarter, days supply lasts into the quarter

if @Date < @MinDate and @NewDate > @MinDate and @NewDate <= @MaxDate

begin

set @ModifiedDaysSupply = DATEDIFF(dd, @MinDate, @NewDate) + 1 –- to avoid 0-indexing the day supply, begins in prior quarter into current quarter

end

-- begins before the quarter, days supply lasts beyond the quarter (should never occur assuming max days supply is 3 months)

else if @Date < @MinDate and @NewDate > @MinDate and @NewDate > @MaxDate

begin

set @ModifiedDaysSupply = DATEDIFF(dd, @MinDate, @MaxDate) + 1

end

-- begins after the quarter, days supply lasts only in the quarter

else if @Date >= @MinDate and @NewDate <= @MaxDate

begin

set @ModifiedDaysSupply = @DaysSupply -- all within the quarter

end

-- begins after the quarter, days supply lasts beyond the quarter

else if @Date >= @MinDate and @Date < @MaxDate and @NewDate > @MaxDate

begin

set @ModifiedDaysSupply = DATEDIFF(dd, @Date, @MaxDate) + 1 -- need to count for it, begins in current quarter and overlaps into next quarter

end

else

begin

set @ModifiedDaysSupply = 0

end

return @ModifiedDaysSupply

end

Application of the above function will convert the Day Supply to the Modified Day Supply. Additionally, a simple CASE statement (below) will provide the Modified Release Date. Altogether, this completes the first processing phase towards your PDC and concomitant utilization with SQL.

, case

when ReleaseDateTime < @StartQuarter then @StartQuarter --start time

else cast(ReleaseDateTime as date)

end as ModifiedReleaseDate

**Table 1. Example of the Prescription Standardization after Processing Step 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Release Date | Day Supply | 🡺 | Modified Release Date | Modified Day Supply |
| 2/1/2018 | 90 | 4/1/2018 | 31 |
| 2/1/2018 | 30 | Null | 0 |
| 4/1/2018 | 60 | 4/1/2018 | 60 |
| 5/1/2018 | 30 | 5/1/2018 | 30 |
| 5/1/2018 | 90 | 5/1/2018 | 61 |

# Step 2 – Stage and Transform the Processed Data

This next portion will require some dynamic SQL to create coverage flags for every single day counted within the timeframe of interest.

Personally, I decided to opt for designating the PDC Calculation to Start with the first day the patient has the medication on hand within the quarter of interest (PDC\_Day0). This was achieved with a simple aggregation on the Patient and Drug (Drug was already limited down to one agent of interest but can be partitioned by the drug as well):

with b as (

select PatientID

, min(ModifiedReleaseDate) over (partition by PatientID) as PDC\_Day0

, ModifiedReleaseDate

, ModifiedDaysSupply

, DateDiff(dd, @StartQuarter, ModifiedReleaseDate) as StartDay

from Test.RxProcessed

where PatientID in (select \* from a)

)

select PatientID

, ModifiedReleaseDate

, ModifiedDaysSupply

, StartDay

, @DaysInTheQuarter - DateDiff(dd, @StartQuarter, PDC\_Day0) as PDCTotalDays

into #Rx\_Step2

from b

**Figure 2. Hardcoded Logic for any single day of coverage**

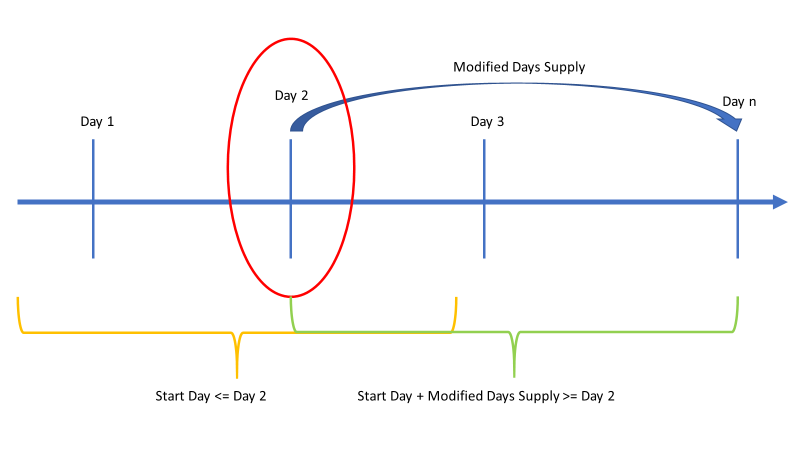


Figure 2 shows the individual case statement needed to flag a single day with the binary 1/0 flag. For the selected day, the prescription MUST be released on or prior to the selected day. Then the addition of the Modified Days Supply to the normalized released date (Start Day) must be greater than or equal to selected day. Meeting these two criteria will indicate that the therapy was on hand during that single day.

“case when StartDay <= @DayNumber and StartDay + ModifiedDaysSupply >= @DayNumber then 1 else 0 end as DayNumber”

Instead of hardcoding this “n” times to span a rolling 365 days or a quarter in my case, use dynamic SQL:

declare @ZeroIndexStart int = 0;

declare @FieldNameIndex int = 0;

declare @FieldName varchar(100);

declare @PDCTotal varchar(max) = ''

declare @SQL1 varchar(max) = '';

while @FieldNameIndex < @DaysInTheQuarter

begin

set @FieldNameIndex += 1;

set @FieldName = concat('D', case when @FieldNameIndex < 10 then '0' else '' end, @FieldNameIndex)

set @PDCTotal += case when @FieldNameIndex = 1 then @FieldName else concat(' + ', @FieldName) end

set @SQL1 += concat(', case when StartDay <=', @ZeroIndexStart, ' and StartDay + ModifiedDaysSupply >= ', @ZeroIndexStart

, ' then 1 else 0 end as ', @FieldName, char(10))

set @ZeroIndexStart += 1

end

drop table if exists Test.RxProcessedStep3;

exec('select \*' + @SQL1 + ' into Test.RxProcessedStep3 from #Rx\_Step2')

With this query, every day within the quarter is categorized using the notation D01-D9x with a corresponding 1/0 flag.

This completes the transformation of the data into the format we can soon manipulate to calculate both PDC and detect concomitant drug therapy days.

# Step 3 – Aggregate the Data

The above currently converts every prescription fill to be expressed as 1/0 over the identified timeframe. But what if the patient had more than one fill of the prescription? That’s currently being structured as a second row of data for the patient by the drug.

Solution: We need to begin aggregating the data to eliminate the fills to coverage by drug.

Very similar to Step 2, the best methodology is to use dynamic SQL to create the query for you rather than needing to hardcode every aggregation.

One major tip in this situation is to use Permanent tables which can be easily referenced by the SSMS’s Information\_Schema to find all fields within a table.

declare @Max varchar(max) = ''

select @Max += concat(', max(', Column\_Name, ') as ', Column\_Name)

from INFORMATION\_SCHEMA.COLUMNS

where Table\_Schema = 'Test' and Table\_Name = 'RxProcessedStep3'

and Column\_Name not in ('PatientID', 'ModifiedReleaseDate', 'ModifiedDaysSupply', 'StartDay')

drop table if exists Test.RxProcessedStep4;

exec('

select PatientID

' + @Max + '

into Test.RxProcessedStep4

from Test.RxProcessedStep3

group by PatientID

'

)

Note: If detecting the presence of concomitant drug therapy, you’ll want to include the drug name here so that the data is retained. For simplicity, I showed an example focused on PDC with 1 drug.

Since my underlying example was using only one Drug, I only grouped by the Patient. This will now yield Patient, D01, D02…D9n.

# Step 4 – Manipulate the Data for your end goal

Now we’re at the end of the project! Do you want to calculate PDC or look for Concomitant Drug Therapy or both? Either way, read on and learn how!

## Proportion Days Covered (PDC)

This is most simply defined by this equation: PDC =

Based on the decisions we made at the very beginning, before Step 1, we can now apply our business rules to complete this calculation.

* # of Days Covered:
* Total # of Days: Must be determined by your choice

Once again, instead of hardcoding the summation of every single day within the timeframe, use dynamic SQL to create the new field based on that n-days you have:

* Please refer to Step 2: @PDCTotal

drop table if exists Test. RxProcessedStep5;

exec('

select ' + @PDCTotal + ' as PDCTotal

, \*

Into Test.RxProcessedStep5

from Test.RxProcessedStep4

')

* Finally, we can apply the equation to the entire dataset:

select PatientID, PDCTotal, PDCTotalDays

, case when PDCTotalDays > 0 then 1.0 \* PDCTotal / PDCTotalDays end as PDC into #PDC

from Test.RxProcessedStep5

PDCTotalDays was identified in Step 2 and incorporated as part of the aggregation in Step 3 making it ready to use in Step 4.

## Concomitant Drug Therapy

This requires including at least another drug so let’s use the assumption that our final table has an additional Drug field.

This dataset will look more like the below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PatientID | Drug | PDCTotalDays | D01 | D02 | D03 | D04 | D05 | DXX |
| A | Sertraline | 91 | 1 | 1 | 1 | 1 | 1 | 1 |
| A | Fluoxetine | 91 | 1 | 0 | 0 | 0 | 0 | 0 |

For concomitant drug therapy, you can first apply some preprocessing to eliminate patients that simply do not meet the number of drugs on hand of interest. For example, if you’re looking at polypharmacy and require a minimum of 5 unique drugs, simply filter the dataset to only those that have at least 5 unique drugs.

This filter can be applied as early as Step 2 but is only optional.

Using the above table, we can easily see that D01 had concomitant therapy between Sertraline and Fluoxetine. Now if the Drug field is cleaned to just the category of agents we want such as antidepressants, we would simply conduct yet another aggregation with a having clause:

declare @HavingClause varchar(max) = ''

declare @Sum varchar(max) = ''

select @Sum += concat(', Sum(', Column\_Name, ') as ', Column\_Name)

, @HavingClause += case

when Column\_Name = 'D01' then concat('sum(', Column\_Name, ') > 1 ')

else concat('OR sum(', Column\_Name, ') > 1 ')

end

from INFORMATION\_SCHEMA.COLUMNS

where Table\_Schema = 'Test' and Table\_Name = 'RxProcessedStep4'

and Column\_Name not in ('PatientID', 'DrugNameWithoutDose')

order by Column\_Name

drop table if exists Test.RxProcessedStep5;

exec ('

select PatientID

' + @Sum + '

Into Test.RxProcessedStep5

from Test.RxProcessedStep4

group by PatientID

having ' + @HavingClause )

Note: The @Sum variable is optional as all you need in this situation is to identify the Patients meeting the criteria. But by keeping the @Sum variable, you can also count how many days within the timeframe did the patient have concomitant drug therapy.

If you made it this far, thanks for reading and good luck with using PDC and/or Concomitant Drug Therapy in both your operations and research projects!