

SQL Query Optimization for Researchers

- As research analysts, we often need costly SQL queries.
 - Wide time ranges
 - Nationwide studies
 - Complicated cohort criteria
- In this presentation, we will talk about how to safely and efficiently approach heavy data needs, and we will troubleshoot some illustrative example queries.
- This is an intermediate level presentation.

“True optimization is the revolutionary contribution of modern research to decision processes.” – George Dantzig



Assumptions

■ I assume:

- You know how to write basic to intermediate SQL queries.
- You know basic CDW architecture, e.g.
 - Dim vs Fact tables
 - Foreign keys
- You can ensure your queries return the correct results.
 - This Cyberseminar is focused on getting the same results faster.
 - If you didn't catch it, check out the recent debugging presentation from Jan 2024 on [VINCI Training & Office Hour \(va.gov\)](#)

The Plan

When we arrive here, we will have a solid foundation to which to attach these details.

Pitfalls!

Tricks!

Tips!

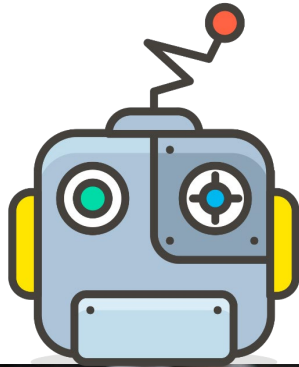
Tree of Knowledge

We will start at the bottom with nontechnical explanations.

Systems and Principles



Meet Your Colleague



Hello
my name is
SQL Query
Optimizer

Strengths:

- Searching
- Sorting
- Syntax
- Arithmetic

Weaknesses:

- Common sense
- Context
- Matching datatypes
- Algebra

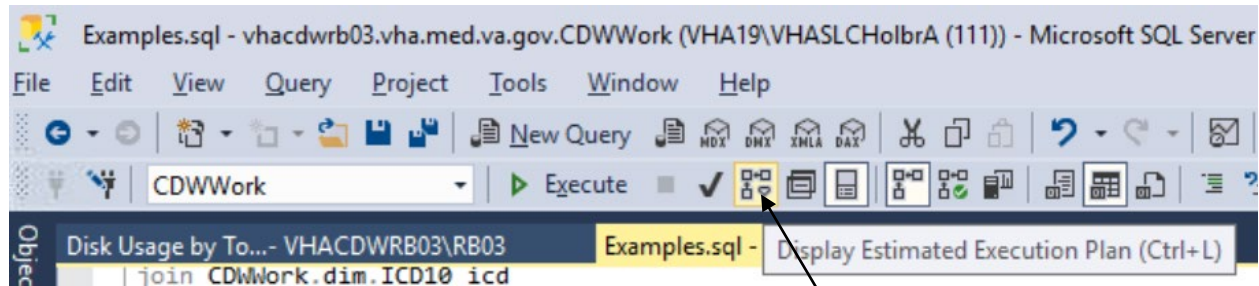
You Are a Team

- Every query you write is a **collaboration** between you and the query engine/optimizer (henceforth “SQL”)
 - You describe what you want
 - Fetch me these records
 - Display them to me like this
 - SQL figures out what to do
 - SQL reads your query to understand what you’re asking for
 - SQL comes up with a plan to fetch what you want
 - SQL passes that plan to the server and displays the result when available

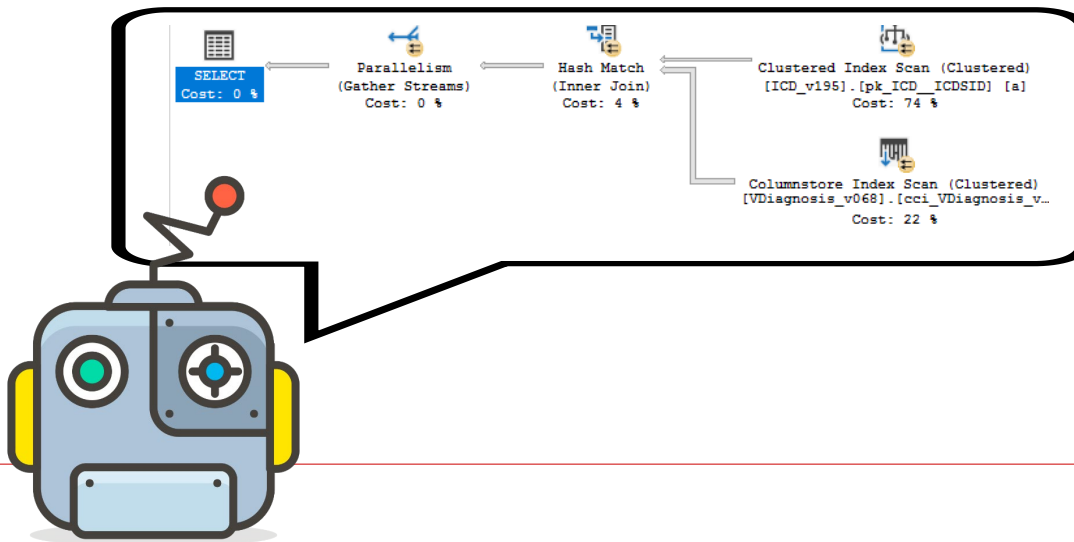
Help Your Teammate

- The division of labor brings great benefits, but there are also downsides:
 - Performance problems are obfuscated
 - Error messages can be unhelpful
- You and your colleague both have strengths and weaknesses
 - Query optimization often comes down to setting your colleague up for success, even if that means getting “out of your lane” sometimes

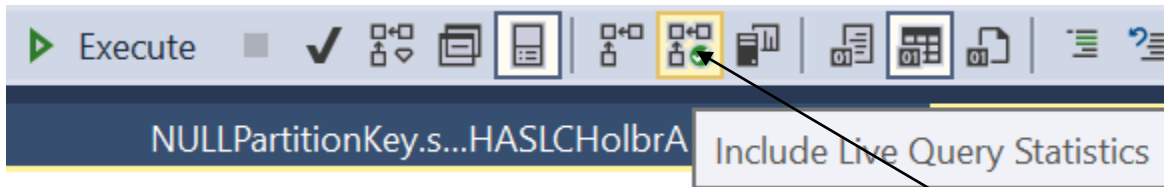
Query Plans



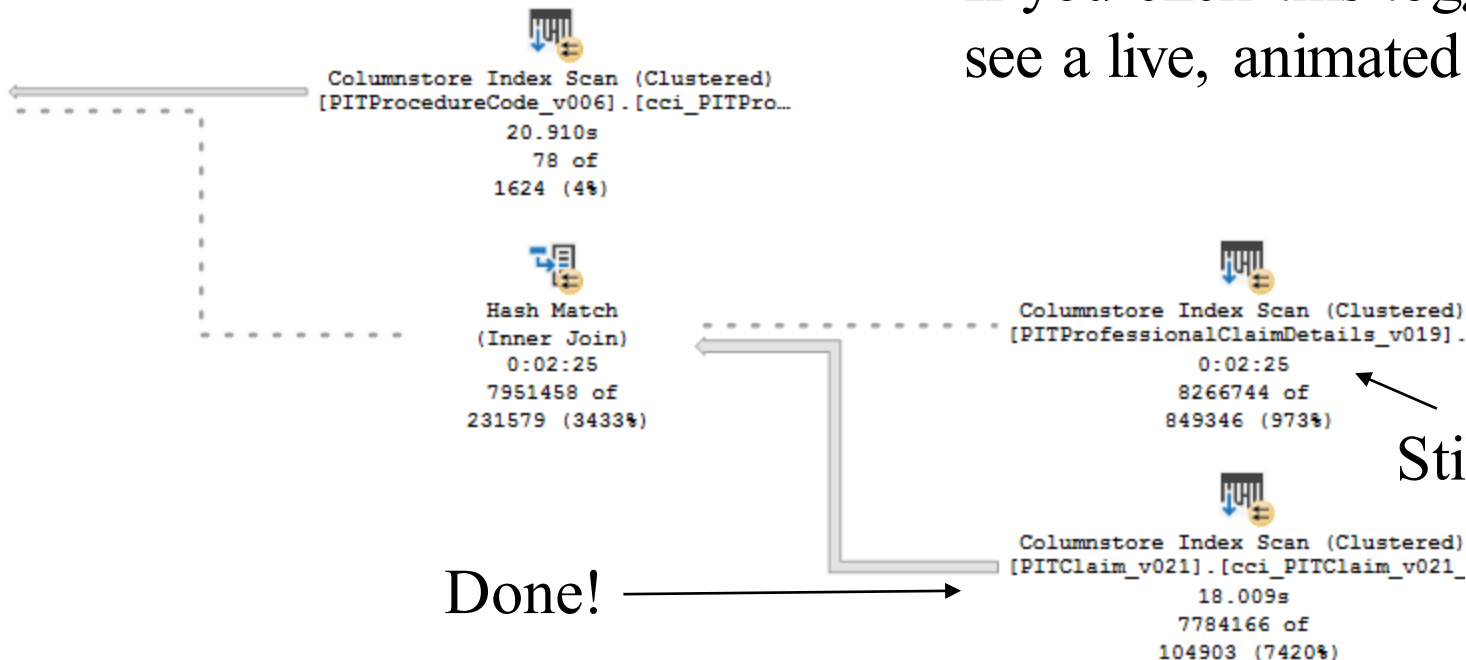
Click! This doesn't run the query, it just asks your colleague what he'd do IF you ran the query.



Live Query Statistics





If you click this toggle, you will see a live, animated execution plan

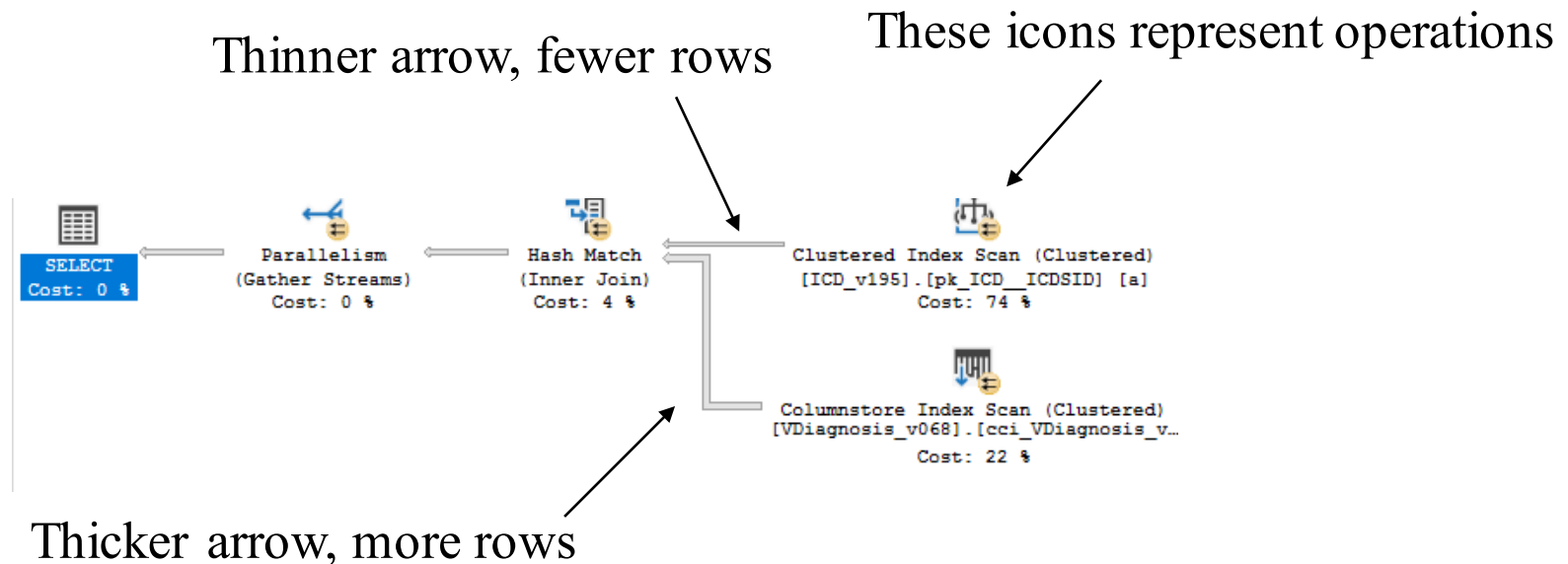


Still working on it

Done!

How to Read a Query Plan

- Each operation passes rows to the next from right to left.
- Mouse over operations for more info.
- Watch for warnings like  and .



More Info from Query Plans

SELECT	
Cached plan size	64 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	76.8856
Estimated Number of Rows	1667.15

Statement

```

select diag.PatientSID
from CDWork.outpat.VDiagnosis diag
join CDWork.dim.ICD10 icd
on diag.ICD10SID = icd.ICD10SID
where diag.Sta3n = 523
and diag.VisitDateTime > convert(
datetime2(0),'2021-01-01')
and icd.ICD10Code like 'B18%'
    
```

This is a running total of the cost

Mouse Over

Relative cost shows which query is more efficient

```

select diag.PatientSID
from CDWork.outpat.VDiagnosis diag
join CDWork.dim.ICD10 icd
on diag.ICD10SID = icd.ICD10SID
where diag.Sta3n = 523
and diag.VisitDateTime > convert(datetime2(0),'2021-01-01')
and icd.ICD10Code like 'B18%'
    
```



```

select diag.PatientSID
from CDWork.outpat.VDiagnosis diag
join #ICD_Hep icd
on diag.ICD10SID = icd.ICD10SID
where diag.Sta3n = 523
and diag.VisitDateTime > convert(datetime2(0),'2021-01-01')
    
```


100 %

Messages Execution plan

Query 1: Query cost (relative to the batch): 77%

```

select diag.PatientSID from CDWork.outpat.VDiagnosis diag join CDWork.dim.ICD10 :
Missing Index (Impact 22.2983): CREATE NONCLUSTERED INDEX [<Name of Missing Index,
    
```

Query 2: Query cost (relative to the batch): 23%

```

select diag.PatientSID from CDWork.outpat.VDiagnosis diag join #ICD_Hep icd on dia
Missing Index (Impact 93.6939): CREATE NONCLUSTERED INDEX [<Name of Missing Index,
    
```



Subtree cost

Cost heuristic:

- Dozens - great, probably just a dim
- Hundreds - it's fine
- Thousands - borderline, but this is how real work gets done
- > 20k - probably bad
- > 300k definitely bad (probably auto-killed)

Mouse over
upper-left icon
SELECT

The screenshot displays a SQL query plan in SQL Server Enterprise Manager. The query is a SELECT statement. The plan shows a Hash Match (Inner Join) operator at the top, which is connected to two Columnstore Index Scan operators. The Hash Match operator has a cost of 0%. The Columnstore Index Scan operator for [Location_v311].[pk_Loc...] has a cost of 46%, and the Columnstore Index Scan operator for [Visit_v224].[cci_Visit...] has a cost of 54%. The Hash Match operator is highlighted with a red box, and its cost details are shown in a tooltip. The tooltip lists the following information:

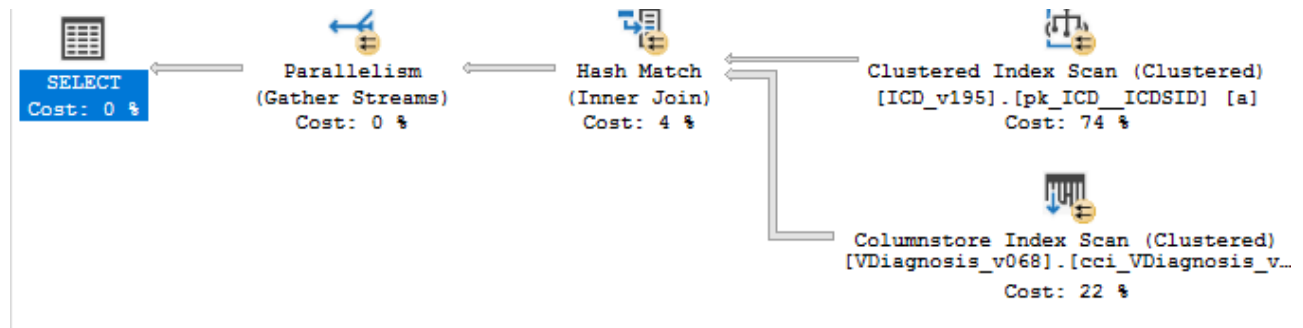
Property	Value
Cached plan size	112 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	111.859
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	9208.49

The Statement text is as follows:

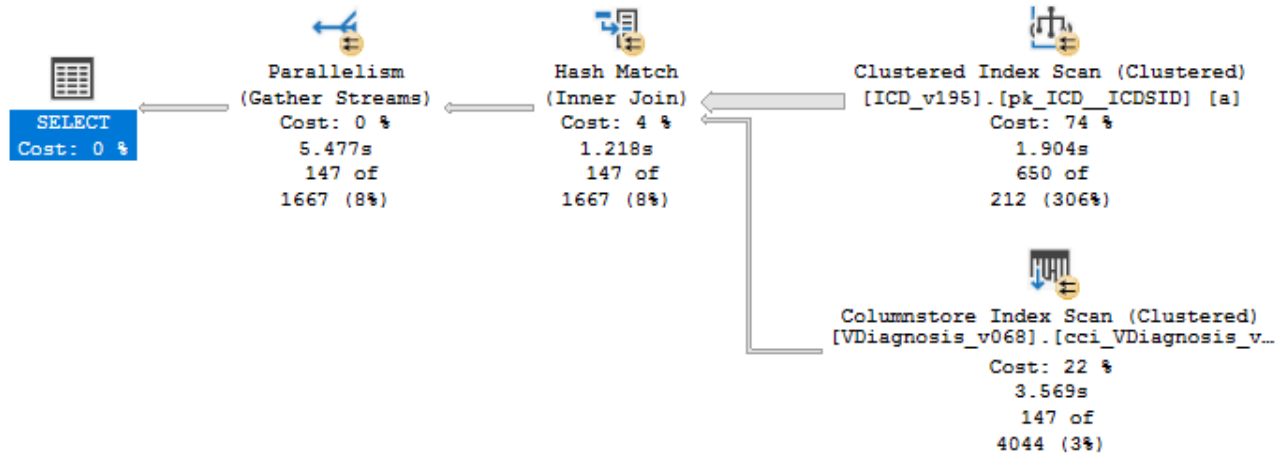
```
SELECT vis.VisitSID, vis.sta3n, vis.VisitDateTime,
vis.PatientSID
,loc.LocationSID, loc.LocationName
FROM ORD_Holbrook_202202042D.Src.Outputpat_Visit vis
JOIN CDWWork.Dim.Location loc
on vis.LocationSID = loc.LocationSID
WHERE vis.VisitDateTime > convert(datetime2(0),'2022-03-
01')
and loc.Sta3n = 548
and loc.LocationType = 'dinic'
```

It was Just an Estimate!

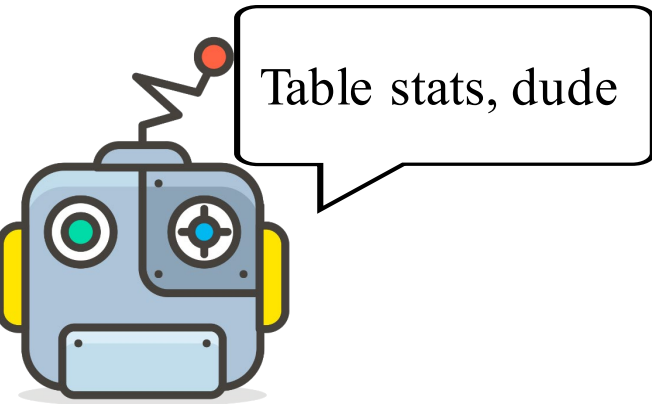
Estimated



Actual



Hey Where Did Those Estimates Come From?



Results		Messages								
Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
codes	Mar 17 2021 9:17AM	73205	73205	177	1	7.440243	YES	NULL	73205	0

All density	Average Length	Columns
1.366027E-05	7.440243	icd10code

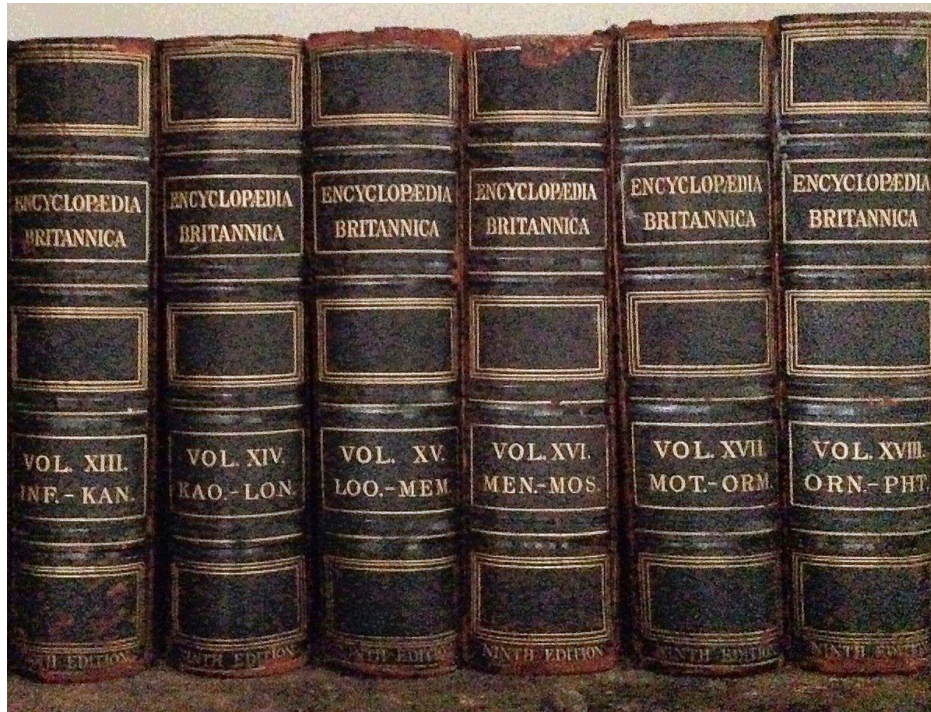
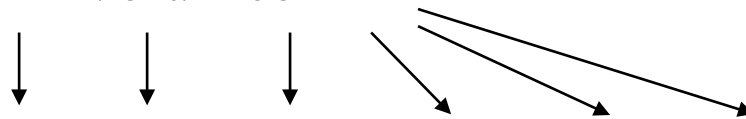
RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
A00.0	0	1	0	1
A69.20	454	1	454	1
B78.9	511	1	511	1
C38.0	255	1	255	1
C7A.012	511	1	511	1
C86.3	383	1	383	1
D12.2	255	1	255	1
D39.0	255	1	255	1
E08.3533	511	1	511	1
E36.01	511	1	511	1
F10.180	383	1	383	1

CDW Partitions

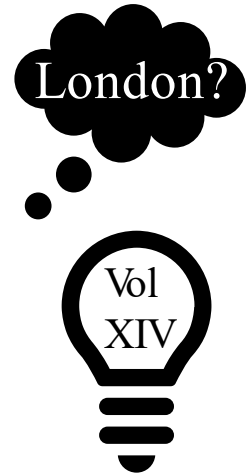
- CDW fact tables are organized (“partitioned”) by date
- Ensuring SQL can skip over (“eliminate”) date ranges it doesn’t need is the most important optimization tool.

Partitions

Volumes

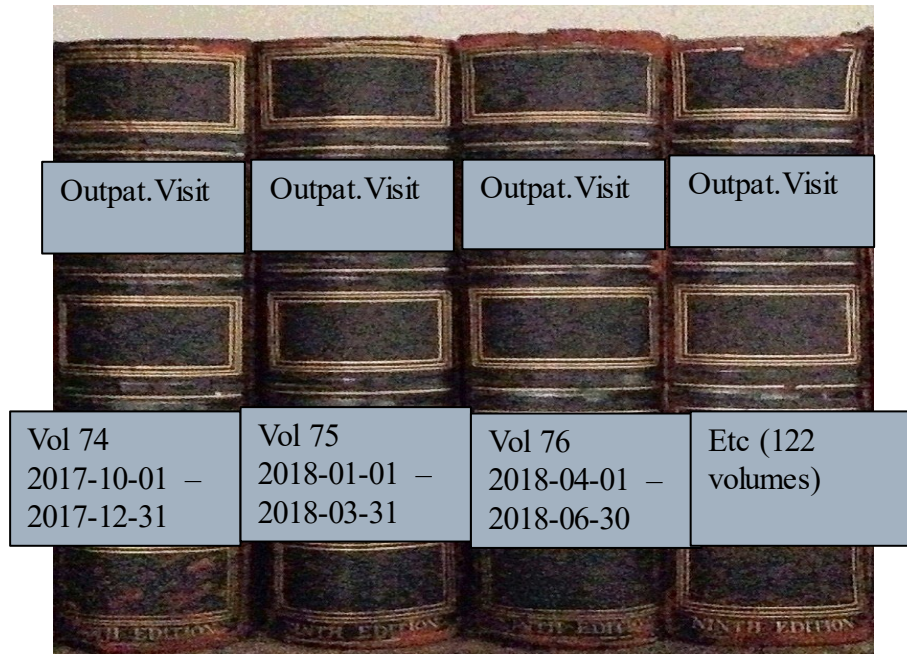


Entries are stored alphabetically, and volumes are labeled with end points



Partitions

Partitions



Entries are stored by date, and partitions are labeled with end points



One More Thing

- Researchers must translate object names when using provisioned views.
 - Best practices, partitioning, indices, and metadata all apply!
 - Your execution plans will have an extra join to your cohort.

[CDWork].[Outpat].[Visit]

[ORD_Ho1brook_202202042D].[Src].[Outpat_Visit]

Now, Finally, Best Practices

- Don't be greedy.
 - Select only the rows and columns you need, especially from fact tables.

```
SELECT *  
FROM  
CDWork.output.Visit  
WHERE...
```

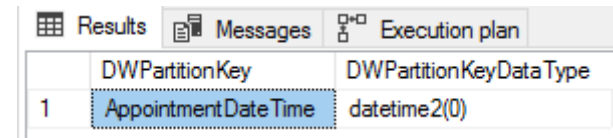
Do you really need all these columns?

Do you really need all these rows?

Partition Elimination

- CDW fact tables are partitioned. In order to use partition elimination:
 - Determine which column is used
 - Check the datatype for that column
 - Use that column and datatype in your WHERE clause

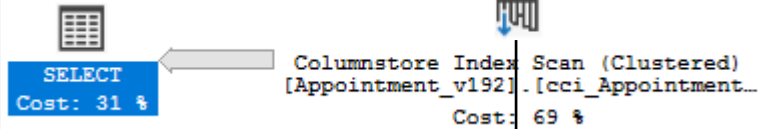
```
SELECT DWPartitionKey, DWPartitionKeyDataType
FROM cdwork.meta.DWView
WHERE DWViewName = 'appointment'
```



	DWPartitionKey	DWPartitionKeyDataType
1	AppointmentDate Time	datetime2(0)

```
SELECT apt.AppointmentSID
FROM CDWork.Appt.Appointment apt
WHERE apt.AppointmentDateTime > convert(datetime2(0), '2021-01-01')
```

Did It Work?



Columnstore Index Scan (Clustered)
Scan a columnstore index, entirely or only a range.

Physical Operation	Columnstore Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Batch
Storage	ColumnStore
Estimated Operator Cost	20.4391 (69%)
Estimated I/O Cost	15.3258
Estimated Subtree Cost	20.4391
Estimated CPU Cost	5.11331
Estimated Number of Executions	1
Estimated Number of Rows to be Read	46434600
Estimated Number of Rows	37205500
Estimated Row Size	22 B
Partitioned	True
Ordered	True
Node ID	1

Predicate
([CDW12].[Appt].[Appointment_v192].[OpCode] as [a],[OpCode]<'X'
OR [CDW12].[Appt].[Appointment_v192].[OpCode] as [a],[OpCode]
>'X') AND ([CDW12].[Appt].[Appointment_v192].[OpCode] as [a],
[OpCode]<'D' OR [CDW12].[Appt].[Appointment_v192].[OpCode] as
[a],[OpCode]>'D') AND [CDW12].[Appt].[Appointment_v192].
[AppointmentDateTime] as [a],[AppointmentDateTime]>'2021-01-01
00:00:00'

Object
[CDW12].[Appt].[Appointment_v192].[cci_Appointment_v192] [a]

Output List
[CDW12].[Appt].[Appointment_v192].AppointmentSID, [CDW12].
[Appt].[Appointment_v192].OpCode

Seek Predicates
Seek Keys[1]: Start: PtnId1001 >= Scalar Operator((87)), End:
PtnId1001 <= Scalar Operator((122))



Use Temp Dims

- Build yourself a temporary dimension with the SIDs you need.
 - Searching with wildcards is fine, because you're only using the dimension.
 - Iterate to your heart's content.
- Don't join to the fact table until you know what you're looking for.
 - In general, wildcard searches on fact tables should be avoided.
- Similarly, you can use temp tables for patient cohorts.

Temp Dim Example

```
SELECT diag.PatientSID
FROM ORD_Holbrook_202202042D.Src.Outputat_Vdiagnosis as diag
JOIN CDWork.dim.ICD10 as icd
  ON diag.ICD10SID = icd.ICD10SID
WHERE diag.Sta3n = 523
  AND diag.VisitDateTime > convert(datetime2(0), '2021-01-01')
  AND icd.ICD10Code LIKE 'B18%'
```

```
SELECT icd10SID
INTO #ICD_Hep
FROM CDWork.dim.ICD10 as icd
WHERE icd.ICD10Code LIKE 'B18%'
```

Just grab the SID
we need for the join

```
SELECT diag.PatientSID
FROM ORD_Holbrook_202202042D.Src.Outputat_Vdiagnosis as diag
JOIN #ICD_Hep as icd
  ON diag.ICD10SID = icd.ICD10SID
WHERE diag.Sta3n = 523
  AND diag.VisitDateTime > convert(datetime2(0), '2021-01-01')
```

Multiple Fact Tables In One Query

- Joining multiple fact tables is not usually recommended.
 - Typically, it is better to refine a cohort over multiple queries.
- If you do use multiple fact tables in one query, remember to use the partition key for each one.

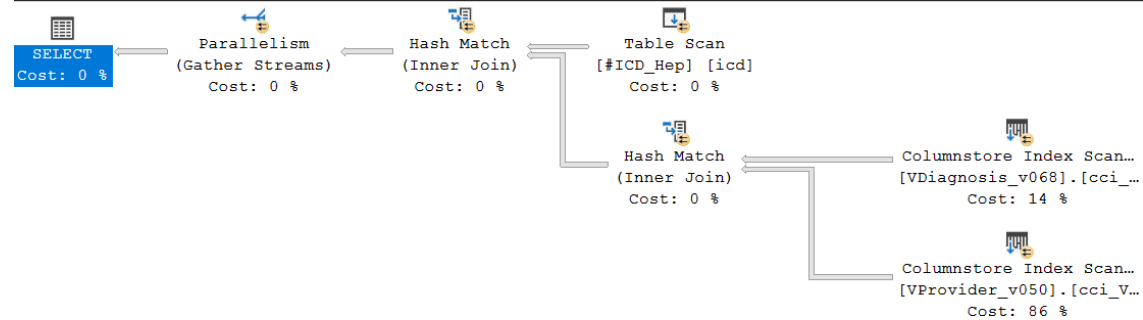
Use Partition Elimination For Each Fact Table

```

select diag.PatientSID, prov.ProviderSID
from CDWork.outpat.VDiagnosis diag
join #ICD_Hep icd
    on diag.ICD10SID = icd.ICD10SID
join CDWork.Output.VProvider prov
    on diag.VisitSID = prov.VisitSID
where diag.Sta3n = 523
    and diag.VisitDateTime
        > convert(datetime2(0), '2021-01-01')
    
```

Query 1: Query cost (relative to the batch): 79%

```
select diag.PatientSID, prov.ProviderSID from CDWork.outpat.VDiagnosis diag join #ICD_Hep icd
```

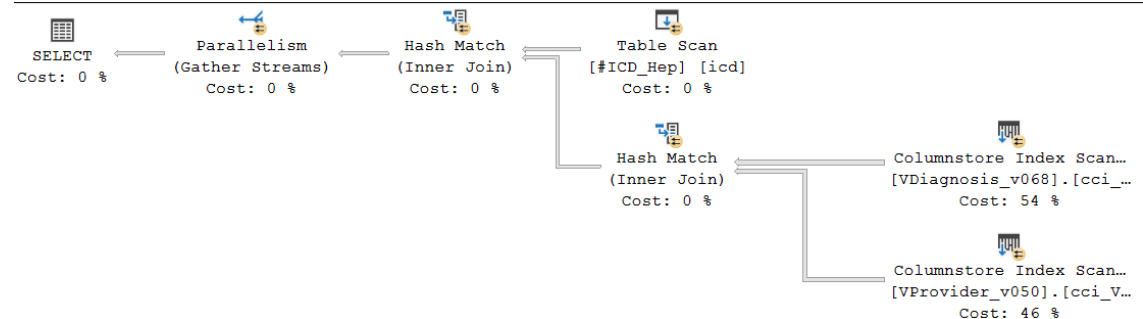


```

select diag.PatientSID, prov.ProviderSID
from CDWork.outpat.VDiagnosis diag
join CDWork.Output.VProvider prov
    on diag.VisitSID = prov.VisitSID
join #ICD_Hep icd
    on diag.ICD10SID = icd.ICD10SID
where diag.Sta3n = 523
    and diag.VisitDateTime
        > convert(datetime2(0), '2021-01-01')
    and prov.VisitDateTime
        > convert(datetime2(0), '2021-01-01')
    
```

Query 2: Query cost (relative to the batch): 21%

```
select diag.PatientSID, prov.ProviderSID from CDWork.outpat.VDiagnosis diag join #ICD_Hep icd
```



Does this change the results?
Nope! But it sure does change the performance.

Functions

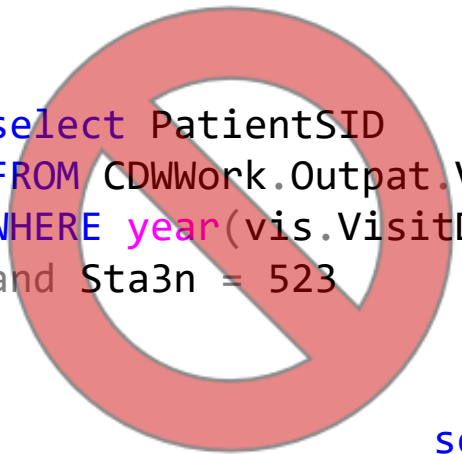
- Avoid using functions on columns in WHERE or JOIN clauses.
 - In the SELECT clause is fine.
- Comparing a column to a function output is the correct way.
- Treat math like a function.
- Bottom line: Columns should be by themselves on one side of the operator in the WHERE clause if at all possible.

Functions on Columns

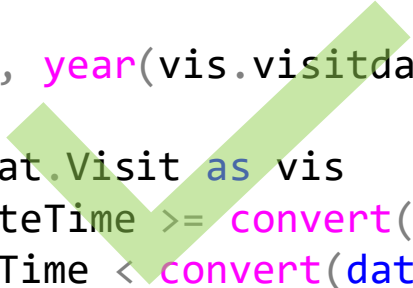
- When comparing columns, you will sometimes need some logic applied to one. In this case:
 - Leave fact table columns as-is, especially the partition column.
 - Pre-filter and pre-calculate in a CTE or temp table.

Column Function Example

- Note the column inside a function in the SELECT clause. This is fine! Just don't do it in a JOIN or WHERE.



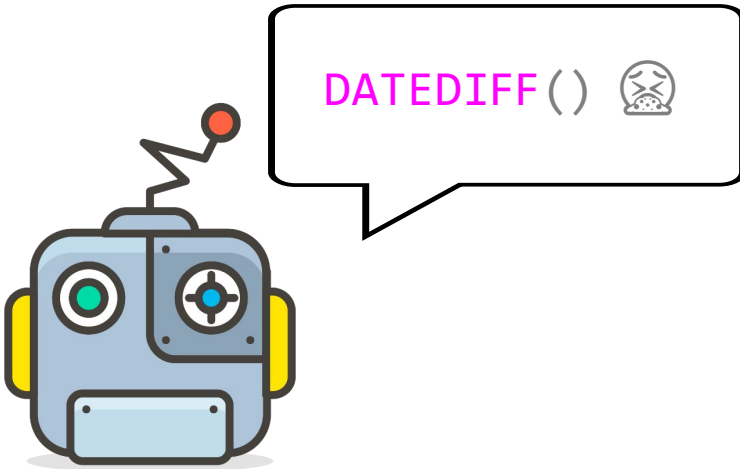
```
select PatientSID
FROM CDWork.Output.Visit as vis
WHERE year(vis.VisitDateTime) = '2020'
and Sta3n = 523
```



```
select PatientSID, year(vis.visitdatetime) as VisitYear
into #temp
From CDWork.Output.Visit as vis
WHERE vis.VisitDateTime >= convert(datetime2(0), '2020-01-01')
and vis.VisitDateTime < convert(datetime2(0), '2021-01-01')
and Sta3n = 523
```

Functions

- Especially avoid functions like DateDiff() with multiple columns used as inputs. You should split the columns so they are on opposite sides of the operator.

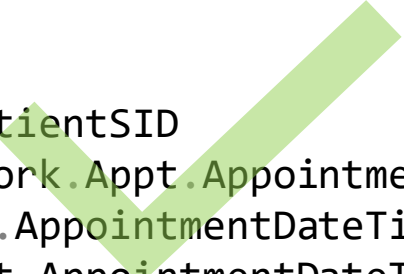


Date Function Example



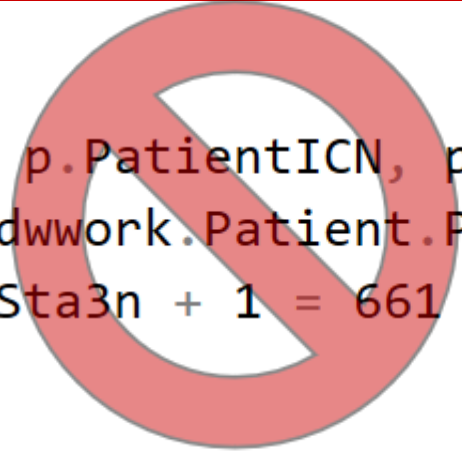
```
SELECT PatientSID
FROM CDWork.Appt.Appointment apt
WHERE datediff(month,apt.AppointmentDateTime ,getdate()) between 0 and 1
```

Also, this yields the wrong result

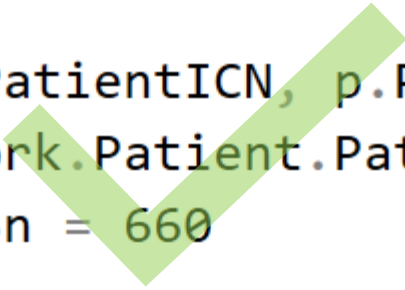


```
SELECT PatientSID
FROM CDWork.Appt.Appointment apt
WHERE apt.AppointmentDateTime >= convert(datetime2(0),dateadd(month,-1 ,getdate()))
AND apt.AppointmentDateTime < convert(datetime2(0),getdate())
```

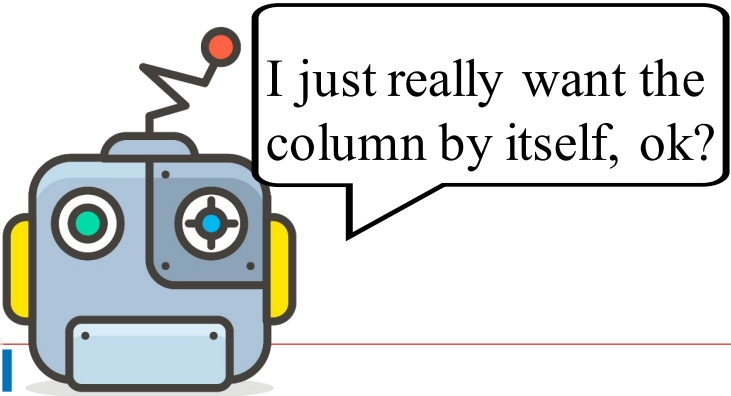
Leave That Column Alone



```
select p.PatientICN, p.PatientSID  
from cdwork.Patient.Patient p  
where Sta3n + 1 = 661
```

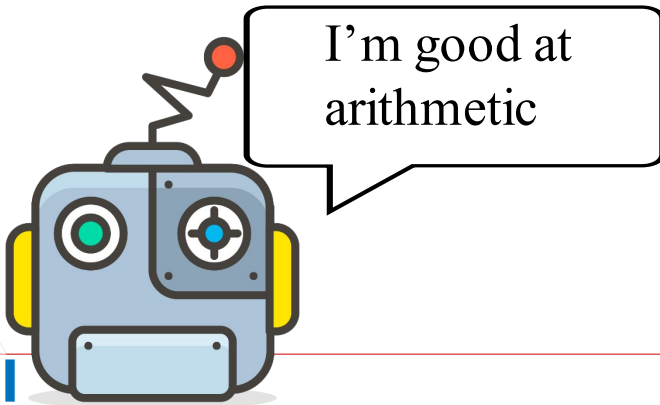


```
select p.PatientICN, p.PatientSID  
from cdwork.Patient.Patient p  
where Sta3n = 660
```



Functions

- Don't worry about functions and math getting complicated. Just make sure it's isolated from the column.



Your Colleague Is Helpful

```
SELECT PatientSID
FROM CDWork.Chem.LabChem lc
WHERE lc.LabChemCompleteDateTime >
      convert(datetime2(0), dateadd(day, (3+2)*-1, getdate()))
```

```
DECLARE @result datetime2(0) = (SELECT dateadd(day, (3+2)*-1, getdate()))
```

```
SELECT PatientSID
FROM CDWork.Chem.LabChem lc
WHERE lc.LabChemCompleteDateTime > @result
```

```
SELECT PatientSID
FROM CDWork.Chem.LabChem lc
WHERE lc.LabChemCompleteDateTime >
      convert(datetime2(0), '2021-03-12 10:54:14.253')
```

Repetition Legitimizes

- Just make sure functions and math are isolated from the columns!

What About NULLs in the Partition Field?

--fine query, but I want to include patients that have not been discharged

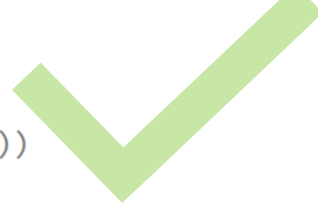
```
select i.InpatientSID, i.PatientSID, i.AdmitDateTime,  
i.DischargeDateTime  
from cdwork.Inpat.Inpatient i  
where i.DischargeDateTime > convert(datetime2(0), '2022-05-01')  
order by i.DischargeDateTime desc
```

What About NULLs in the Partition Field?

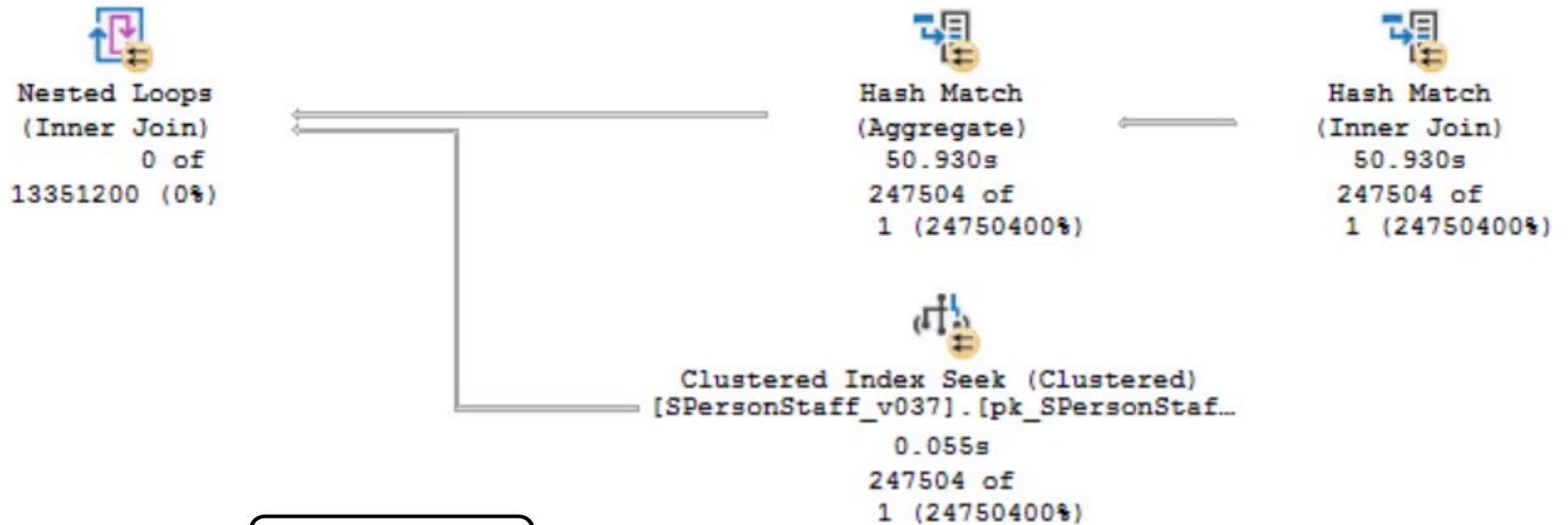
```
--no good, column inside function
--plus there are some weird old records where discharge never got filled in
select i.InpatientSID, i.PatientSID, i.AdmitDateTime, i.DischargeDateTime
from cdwork.Inpat.Inpatient i
where isnull(i.DischargeDateTime, getdate()) > convert(datetime2(0), '2022-05-01')
order by i.DischargeDateTime desc
```

```
--no good, this use of OR prevents SQL from using the partition key
--humans know discharges should be after admissions, but SQL doesn't know that
select i.InpatientSID, i.PatientSID, i.AdmitDateTime, i.DischargeDateTime
from cdwork.Inpat.Inpatient i
where (i.DischargeDateTime > convert(datetime2(0), '2022-05-01')
      or i.AdmitDateTime > convert(datetime2(0), '2022-05-01'))
order by i.DischargeDateTime desc
```

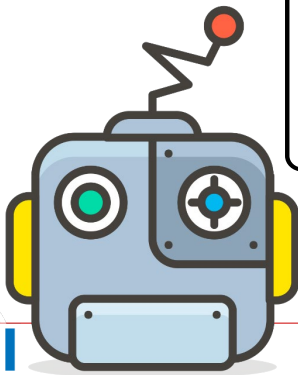
```
--this is the way
--explicitly tell SQL which partitions to look in for each set of conditions, either with a date range or NULL
select i.InpatientSID, i.PatientSID, i.AdmitDateTime, i.DischargeDateTime
from cdwork.Inpat.Inpatient i
where i.DischargeDateTime > convert(datetime2(0), '2022-05-01')
      or (i.DischargeDateTime is null and i.AdmitDateTime > convert(datetime2(0), '2022-05-01'))
order by i.DischargeDateTime desc
```



Whoa, Bad Estimates!




Whoops.



Bad Estimates

- Watch out for nested loops, which are only good for small amounts of rows



```
Nested Loops  
(Inner Join)  
0 of  
13351200 (0%)
```

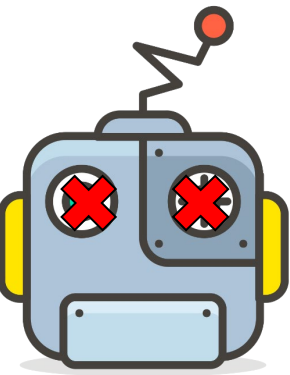
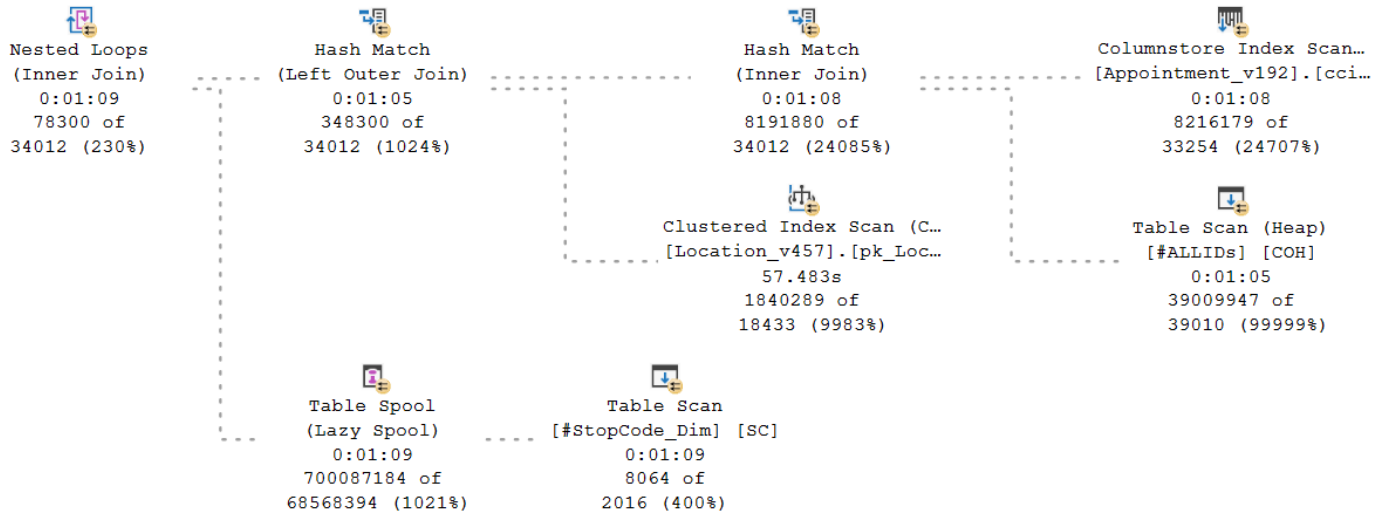
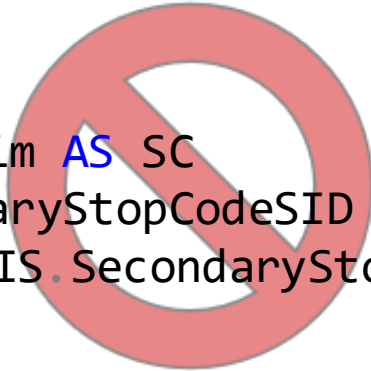
- Try adding a predicate on an indexed column like Sta3n.
- Look for columns inside functions and bad joins

Don't Use OR In Your Joins

- It is correct to try to hit fact tables fewer times - preferably just once
- **However** it is a bad practice to use an OR in your join
- This is common in two scenarios:
 - Primary and secondary stop codes
 - ICD 9 and 10 codes
- Instead, use a union
 - Hitting the fact table twice with good performance is much better than once with bad performance

This Query Gets Killed

...
 INNER JOIN
 #StopCode_Dim AS SC
 ON VIS.PrimaryStopCodeSID = SC.StopCodeSID
 OR VIS.SecondaryStopCodeSID = SC.StopCodeSID
 ...



Use a Union Instead

```
...  
INNER JOIN  
#StopCode_Dim AS SC  
ON VIS.PrimaryStopCodeSID = SC.StopCodeSID  
...  
UNION  
...  
INNER JOIN  
#StopCode_Dim AS SC  
ON VIS.SecondaryStopCodeSID = SC.StopCodeSID  
...
```

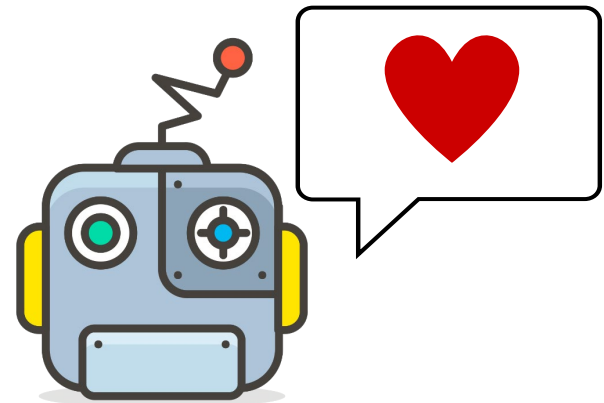


Query Hints

- In general, **don't** use query hints.
 - Remember, the division of labor with your colleague is a good thing (“stay in your lane”)
- **However**, if you can't convince SQL that there will be many rows, you can **as a last resort** force a hash match instead of a nested loop: `INNER HASH JOIN`

Key Takeaways

- Don't be greedy!
- Use partitions!
- Separate functions from columns.
- Set your colleague up for success.



Final Thoughts

- “Premature Optimization is the root of all evil” - Donald Knuth
 - You should always use the best practices of not being greedy and using partition elimination, etc.
 - Beyond that, spend your brain power getting the right results, rather than squeezing a 1 minute query down to 30 seconds.
- Query optimization is difficult
 - Ask for help!

VINCI Resources

- [VINCI University \(va.gov\)](#)
- [VINCI Training & Office Hour \(va.gov\)](#)
 - VINCI Office Hours every Wednesday at 3 PM ET.
- [Managing Research Data](#) from last month

OIT CDW Resources

- Data Services & Field Support (sharepoint.com)
 - Especially Six Simple Steps and Easy Eight
- SQL Training (sharepoint.com)

One More Resource

- [SQL Server Execution Plans, Third Edition, by Grant Fritchey - Simple Talk \(red-gate.com\)](#)

Acknowledgements

- VINCI Data Services
- CDW Data Services and Field Support
- VINCI SQL and CDW Bootcamp Alumni

Questions?

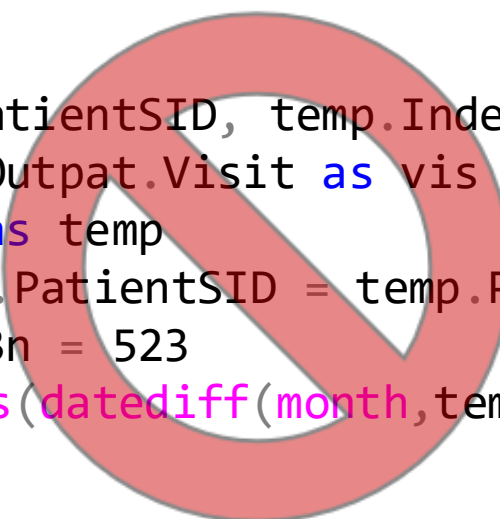
- And if you think of more questions later:
 - [VINCI Training & Office Hour \(va.gov\)](#)
 - VINCI Office Hours every Wednesday at 3 PM ET.
 - VINCI@va.gov
- A bonus example we don't have time for in subsequent slides ↓

Appendix: A Tricky Problem

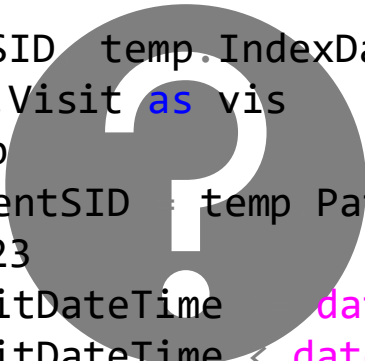
- How can I use partition elimination when I'm comparing to an index date, i.e. I have a dynamic date range?
- The simple answer: If you have any kind of study window or anything you can use, just use that, even if it's wide.
- Otherwise...

Appendix Example


```
select temp.PatientSID, temp.IndexDate, vis.VisitDateTime
FROM CDWork.Output.Visit as vis
JOIN #visits as temp
    on vis.PatientSID = temp.PatientSID
WHERE vis.Sta3n = 523
    and abs(datediff(month,temp.IndexDate, vis.VisitDateTime)) <= 1
```



```
select temp.PatientSID temp.IndexDate, vis.VisitDateTime
FROM CDWork.Output.Visit as vis
JOIN #visits as temp
    on vis.PatientSID = temp.PatientSID
WHERE vis.Sta3n = 523
    and vis.VisitDateTime > dateadd(month, -1, temp.IndexDate)
    and vis.VisitDateTime < dateadd(month, 1, temp.IndexDate)
```



We replaced
datediff() with
dateadd()...but will
SQL use the
partition with that
index date in the
function?



Did It Work?

Columnstore Index Scan (Clustered)
Scan a columnstore index, entirely or only a range.

Physical Operation	Columnstore Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Batch
Storage	ColumnStore
Estimated Operator Cost	1177.38 (100%)
Estimated I/O Cost	1074.18
Estimated Subtree Cost	1177.38
Estimated CPU Cost	103.201
Estimated Number of Executions	1
Estimated Number of Rows	35223.4
Estimated Number of Rows to be Read	3752060
Estimated Row Size	20 B
Partitioned	True
Ordered	False
Node ID	5

Predicate
[CDW15].[Output].[Visit_v224].[OpCode] as [a],[OpCode]<>'X' AND [CDW15].[Output].[Visit_v224].[OpCode] as [a],[OpCode]<>'D' AND [CDW15].[Output].[Visit_v224].[Sta3n] as [a],[Sta3n]=(523) AND PROBE ((Opt_Bitmap1009],[CDW15].[Output].[Visit_v224].[PatientSID] as [a],[PatientSID]))

Object
[CDW15].[Output].[Visit_v224].[cci_Visit_v224] [a]

Output List
[CDW15].[Output].[Visit_v224].VisitDateTime, [CDW15].[Output].[Visit_v224].PatientSID, [CDW15].[Output].[Visit_v224].OpCode

Query executed successfully.

Ready
Tue 3/24 PM
PS



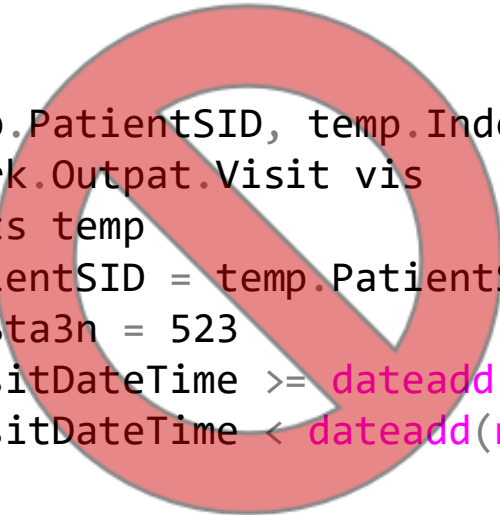
Seek Predicates
Seek Keys[1]: Start: PtnId1001 >= Scalar Operator(RangePartitionNew ((@mindate],[1], '1999-10-01 00:00:00','2000-01-01 00:00:00','2000-04-01 00:00:00','2000-07-01 00:00:00','2000-10-01 00:00:00','2001-01-01 00:00:00'))



Why Not?

- If partition elimination is used, it will be the first step.
- Therefore, it can't be used with something that needs to be evaluated for every row.

```
select temp.PatientSID, temp.IndexDate, vis.VisitDateTime
FROM CDWork.Outpat.Visit vis
JOIN #visits temp
on vis.PatientSID = temp.PatientSID
WHERE vis.Sta3n = 523
and vis.VisitDateTime >= dateadd(month, -1, temp.IndexDate)
and vis.VisitDateTime < dateadd(month, 1, temp.IndexDate)
```

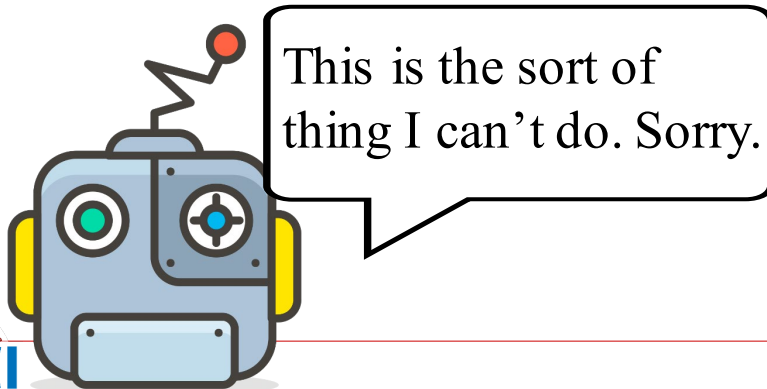


Should We Use the Partition?

```
select min(dateadd(month, -1, IndexDate)) FROM #visits
--2017-12-01 00:00:01
select max(dateadd(month, 1, IndexDate)) FROM #visits
--2018-02-01 23:59:54
```

Look at that narrow range! We SHOULD use the partition...but how?

A good plan would use the max and min dates to grab the right partition.



Let's Help Out

```
create clustered index idx_date on #visits (PatientSID,indexdate) --didn't help
```

```
create statistics dates on #visits (indexdate) --didn't help
```

```
select temp.PatientSID, temp.IndexDate, vis.VisitDateTime
FROM CDWork.Output.Visit vis
JOIN #visits temp
on vis.PatientSID = temp.PatientSID
WHERE vis.Sta3n = 523
and vis.VisitDateTime >= dateadd(month, -1, temp.IndexDate)
and vis.VisitDateTime < dateadd(month, 1, temp.IndexDate)
and vis.VisitDateTime >= (select min(dateadd(month, -1, IndexDate)) FROM
#visits) --didn't help
```

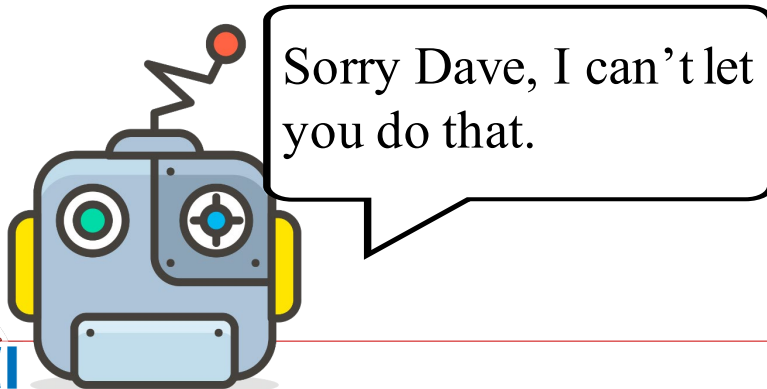
Why Didn't That Last Try Work?

```
vis.VisitDateTime >= (select min(dateadd(month, -1, IndexDate)) FROM #visits)
--didn't help
```

If IndexDate was replaced with getdate(), then SQL would be able to use partition elimination. Even though select min()... will obviously only return one value, SQL doesn't know to do that part first.

Do That Part First, Please

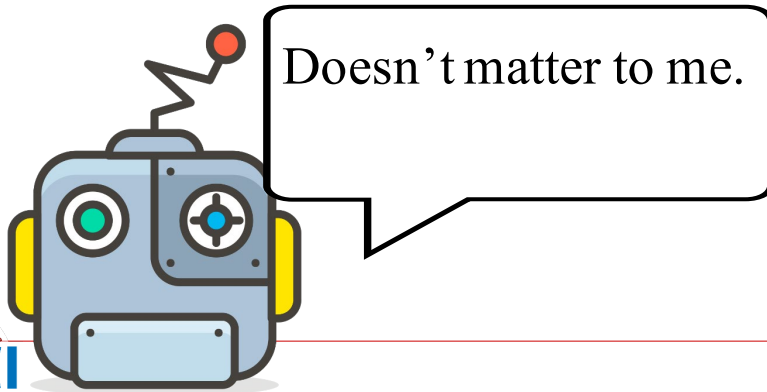
- How about if we move that line up to be the first thing in the WHERE clause?
- How about if we change the JOIN order to look at the temp table first?
- How about if we move that WHERE criterion into the inner JOIN clause?



SQL decides for itself all of the above. Changing the order in your query makes no difference to the execution plan.

So Order Doesn't Matter?

- Yes it does! Remember that your script should be readable for you and other humans.



Back to the Problem

```
declare @mindate datetime2(0) =  
    (select dateadd(month, -1, min(IndexDate))  
     FROM #visits)
```

```
declare @maxdate datetime2(0) =  
    (select dateadd(month, 1, max(IndexDate))  
     FROM #visits)
```

This is one of the few cases where I think variables are fine

```
select temp.PatientSID, temp.IndexDate, vis.VisitDateTime  
FROM CDWork.Outpat.Visit vis  
JOIN #visits temp  
    on vis.PatientSID = temp.PatientSID  
WHERE vis.Sta3n = 523  
    and vis.VisitDateTime >= dateadd(month, -1, temp.IndexDate)  
    and vis.VisitDateTime < dateadd(month, 1, temp.IndexDate)  
    and vis.VisitDateTime >= @mindate --eureka  
    and vis.VisitDateTime < @maxdate --I have found it
```