Weighted Averaging with Apache Hive and SQL Server 2008 R2

May 24, 2012

By J'son Cannelos
Partner/Principal Architect

To Be Is To Do - *Shakespeare*
To Do Is To Be - *Voltaire*
DoBeDoBeDo - *Sinatra*

I recently had an interesting challenge with a client who wanted to see how long a particular group of users would watch an online video. Through a partnership with Video Content leader, Freewheel (http://www.freewheel.tv/), our client knew exactly when a user hit a particular online video, how long they stayed there, and for good measure, that the user belonged to a group of segments that further defined who they were. A user could belong to segments such as "College Men" or "Mothers", et. In all, there were more than 100 segments available and user could belong to more than one (although cross over was rare).

Due to the sheer amount of log data available for the client, Apache Hadoop was the only reasonable way to store and backup this data while still allowing for reasonably quick analysis. Installed with the cluster was Apache Hive – a data warehouse system that allows for ad-hoc queries and more – no Java required! See Raul Olvera's recent article, "Starting with Hive" (http://esagegroup.wordpress.com/2012/05/17/starting-with-hive/), for a good primer on this subject.

It took some work in Hive to get the data in a state agreeable for averaging. Since this topic could take up an entire new post, we'll skip ahead for now.

My final output from my Hive queries was a simple text file containing a record for each transaction that occurred in a month, along with the transaction's segment id and duration spent in seconds on a video.

12/1/2025	15	34.654
12/1/2025	16	2
12/1/2025	15	100.53
12/1/2025	9	341.5

NOTE: Dates and results have been changed.

And on and on it went. My final text output was 1.6 GB! Over 96 Million rows. :P I had wanted to use Apache Sqoop (http://sqoop.apache.org/) to move the data directly into my SQL Server database, but it wasn't installed yet. So yes, I had to copy the entire text file down to my local box, and then use SQL Management Studio (Tasks à Import Data) to import it into a new table in my database. I did add an IDENTITY (1, 1) field to my table so I would have a way of telling records apart – this will be important later! For my data, my table schema was thus:

```
if object_id('FactSegment') is not null
	drop table FactSegment
go

create table FactSegment
(
	SegmentDurID int not null identity(1, 1) primary key,
	RequestDate smalldatetime not null,
	SegmentID int not null,
	SecsDuration int not null
)

alter table FactSegment
	add constraint fk_FactSegment_DimSegment foreign key (SegmentID) references DimSegment
(SegmentID)
```

DimSegment was my lookup table, connecting segment ids to actual names. When dealing with a lot of data, normalizing tables like this will save tons of space, but increase query / lookup time. In any event, I had my data loaded by date, segment, and duration.

There were a lot of durations with only 1 second – someone opening a video and leaving immediately. I didn't want them to skew the overall average. Also, I wanted to group my "weighted" average by segment and day. This meant removing the top and bottom X% of transactions, according to SecsDuration, when grouped by RequestDate and SegmentID. An interesting challenge!

With 96 Million rows some pre-aggregation was in order. First of all, what if I could "mark" the transactions that fell into the top and bottom X%? Then I could just make sure those records were excluded when doing my averaging queries and we'd be good to go!

First, I created an Exclusion table:

```
if object_id('ExcludeSegDurRecords') is not null
	drop table ExcludeSegDurRecords

create table dbo.ExcludeSegDurRecords (SegmentDurID int not null primary key)
```

All it does is keep track of transactions that I don't want included in my averaging. Next, since I want my top and bottom X% to be excluded at the Date and Segment level, I needed a cursor.

I know, I know... cursors are bad, evil things that lurk in the dark and TP your house when you're away. A SQL guru whose name escapes me always signed off his blogs with "Cursors are useful if you don't know SQL". J I respectfully disagree, especially if you don't have all day to come up with that one uber SQL query that will get all your work done at once. The start of my cursor is pretty simple:

```
-- cursor to update records that should be excluded from weighted average. ids recorded in new
table.
declare       @requestDate smalldatetime
declare       @segmentID int

declare Looper cursor
for
       select distinct
              RequestDate,
              SegmentID
       from   FactSegment
       order by 1, 2

open Looper

fetch next from Looper into @requestDate, @segmentID
```

At this point, it's a pretty straightforward cursor. It takes every combination of RequestDate and SegmentID in my FactSegment table and assigns them to variables, one at a time. Then it starts its loop.

```
while @@fetch_status = 0
begin
       print 'Date = ' + convert(varchar(30), @requestDate, 110) + '; Segment = ' + convert(varchar
(20), @segmentID)

insert into ExcludeSegDurRecords (SegmentDurID)
select distinct SegmentDurID
from
(
       select SegmentDurID,
              row_number() over (order by secsDuration) rn,                                  count(*)
over () countRows
       from   FactSegment
       where  SecsDuration > 0
       and    RequestDate = @requestDate
       and    SegmentID = @segmentID
       ) N
where rn not between countRows * 0.05 and countRows * 0.95         -- get ids of outliers for day
and segment!
       --where rn between countRows * 0.05 and countRows * 0.95

       fetch next from Looper into @requestDate, @segmentID
end

close Looper
deallocate Looper
```

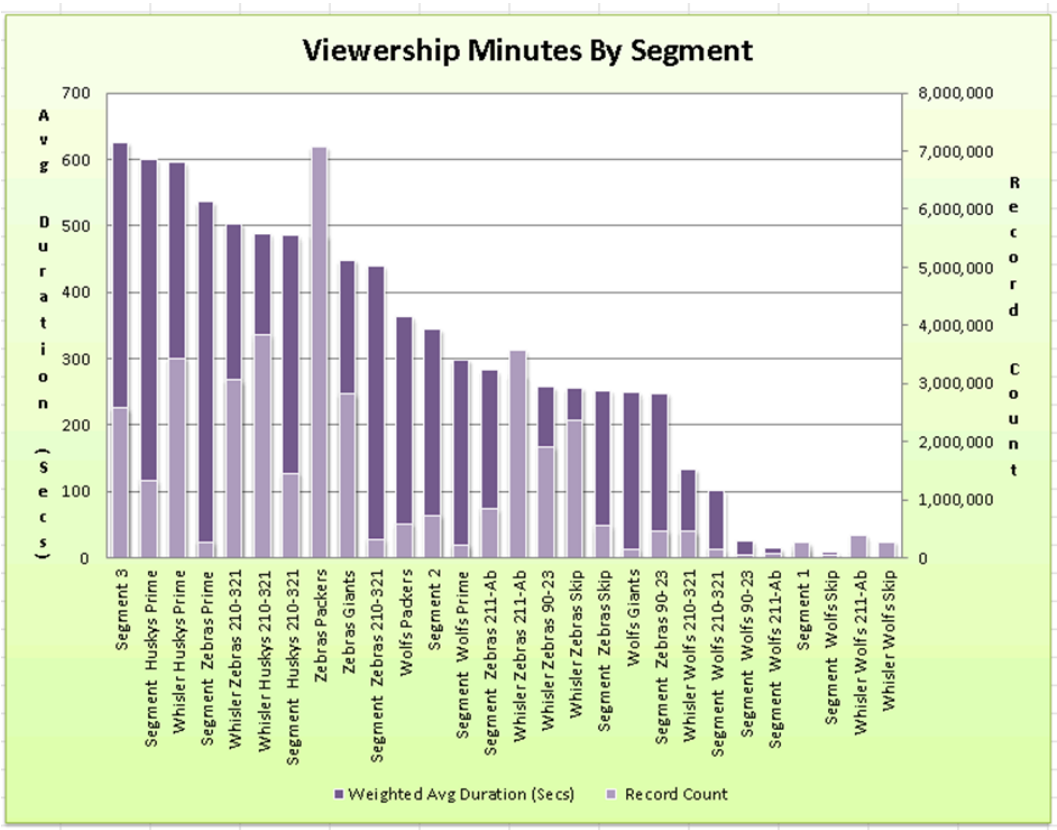
While we're in the loop, this is where the magic happens. Remember, I am recording the transactions (SegmentDurID) of records whose secsDuration is in the top or bottom 5% of all records for **that day and segment**. `count(*) over ()` gives me an overall count of records for that day and segment (note to self: over() may be optional here). `row_number() over (order by secsDuration)` is more interesting. This is essentially getting the row number for each row, but as ordered by secsDuration! This means that row number 1, 2, 3, 4 and so on will be all my lowest durations (like 1 second), and the very final row numbers will be the highest duration. This is very useful. All the outer where clause has to do - `where rn not between countRows * 0.05 and countRows * 0.95` – is to limit the inner query to only those records whose row number is outside of the 5% - 95% range! These specific records are added to my ExcludeSegDurRecords table for later.

The cursor took about an hour to run – the SQL Server is on a low horsepower dev box. J One thought – why not just update an "exclude" bit field in my main table? Due to the size of the Fact table, my cursor would have taken days! Most updates against large tables are very time consuming and need to be done with care. Indexes get rebuilt, constraints checked, and on and on. Putting the ids in a new table was the correct design decision.

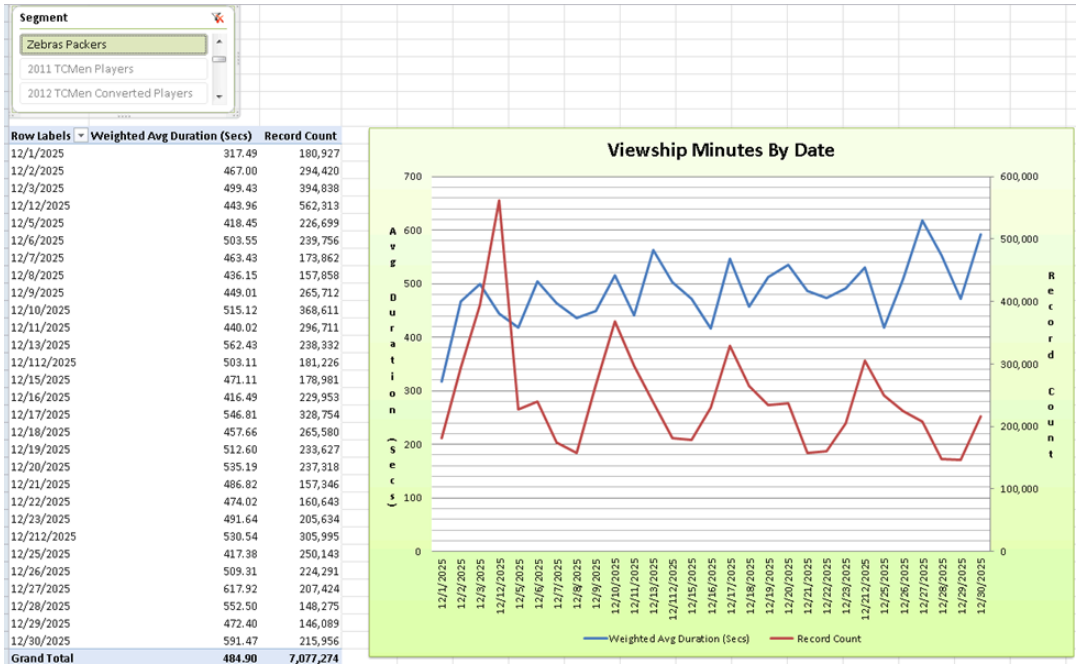
So my outliers were captured, now I just need to write an averaging query in SQL:

```
select fs.RequestDate,
       fs.SegmentID,
       s.Segment,
       s.SegmentGroup,
       count(*) as [Record Count],
       avg(convert(decimal(12, 6), fs.SecsDuration)) as [Weighted Average Duration (Secs)]
from   FactSegment fs
join   DimSegment s on     fs.SegmentID = s.SegmentID
left join ExcludeSegDurRecords e on     fs.SegmentDurID = e.SegmentDurID
where  e.SegmentDurID is null
group by fs.RequestDate, fs.SegmentID, s.Segment, s.SegmentGroup
having count(*) > 1
order by 1, 4, 3
```

I copied these results into a quick Pivot Table in Excel and I was able to have some fun:



Segments and actual results have been changed to protect the innocent, but you get the idea. This is a grouping by Segment across an entire month.



Row Labels	Weighted Avg Duration (Secs)	Record Count
12/1/2025	317.49	180,927
12/2/2025	467.00	294,420
12/3/2025	499.43	394,838
12/12/2025	443.96	562,313
12/5/2025	418.45	226,699
12/6/2025	503.55	239,756
12/7/2025	463.43	173,862
12/8/2025	436.15	157,858
12/9/2025	449.01	265,712
12/10/2025	515.12	368,611
12/11/2025	440.02	296,711
12/13/2025	562.43	238,332
12/112/2025	503.11	181,226
12/15/2025	471.11	178,981
12/16/2025	416.49	229,953
12/17/2025	546.81	328,754
12/18/2025	457.66	265,580
12/19/2025	512.60	233,627
12/20/2025	535.19	237,318
12/21/2025	486.82	157,346
12/22/2025	474.02	160,643
12/23/2025	491.64	205,634
12/212/2025	530.54	305,995
12/25/2025	417.38	250,143
12/26/2025	509.31	224,291
12/27/2025	617.92	207,424
12/28/2025	552.50	148,275
12/29/2025	472.40	146,089
12/30/2025	591.47	215,956
Grand Total	**484.90**	**7,077,274**

And this chart shows how Average Duration and overall Log counts have compared over the same month for the Zebra Packers Segment. The Segment at the top is a Slicer, a useful and new Pivot Table feature in Excel 2010.

Anyways, that's how we got this done. Another day, I
'll do a Hive specific post on this assignment. Anyone for Hive User Defined Functions? :P