Computed columns

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:32 AM PST

Contents

Introduction

The following TSG is intended to give a summarized view of what is a computed column

What is a computed column

A computed column is a virtual column that contains a value that is computed by an expression.

This value can be stored (PERSISTED) or not (NON PERSISTED).

An example of an computed column - you want a column that shows the sum of two values of two different columns on the same row.

Persisted

The computed value is persisted in storage. Since it's stored, it can be used as Primary Key, Foreign Key or even be indexed.

Not Persisted

It´s not saved in storage, like so the value is calculated when a select is executed on a table.

Demo - Non persisted computed column

First, let´s start with a non persisted computed column.

We will create a table with two columns - col1 and col2. The third column (computed column col3) is the result of the sum of col1 and col2. We will also insert some rows:

```
create table table1(
col1 int,
col2 int,
col3 as col1 + col2
)
go

insert into table1 values (1,2)
insert into table1 values (2,3)
insert into table1 values (4,5)
insert into table1 values (6,7)
insert into table1 values (8,9)
go
```

When selecting from this table the value for col3 is computed:

```
select * from table1



col1        col2        col3
----------- ----------- -----------
1           2           3
2           3           5
4           5           9
6           7           13
8           9           17
```

Looking at the execution plan, we can see the computation (means that the value for col3 is always computed on each select - check *Compute Scalar(DEFINE:([archetype_rimarqu].[dbo].[table1].[col3]=[archetype_rimarqu].[dbo].[table1].[col1]+[archetype_rimarqu].[dbo].[table1].[col2]*):

```
SET SHOWPLAN_TEXT ON
go
select * from table1
GO
SET SHOWPLAN_TEXT OFF
GO



StmtText
---------------------
select * from table1

(1 row affected)

StmtText
-----------------------------------------------------------------------------------------------------------------
  |--Compute Scalar(DEFINE:([archetype_rimarqu].[dbo].[table1].[col3]=[archetype_rimarqu].[dbo].[table1].[col3
       |--Compute Scalar(DEFINE:([archetype_rimarqu].[dbo].[table1].[col3]=[archetype_rimarqu].[dbo].[table1].
            |--Table Scan(OBJECT:([archetype_rimarqu].[dbo].[table1]))
```

A non persisted computed column, despite of saving storage might have a performance concern since it has to compute the value for each row.

Demo - Persisted computed column

First let's drop the previous table and recreate with the same definition and values. The only difference is that the computed column is now persisted:

```
--Drop previous demo table
DROP table table1
go

create table table1(
col1 int,
col2 int,
col3 as col1 + col2 PERSISTED
)
go

insert into table1 values (1,2)
insert into table1 values (2,3)
insert into table1 values (4,5)
insert into table1 values (6,7)
insert into table1 values (8,9)
go
```

If we check the execution plan for the select of the whole table, now the Computation of the value is no longer needed:

```
SET SHOWPLAN_TEXT ON
go
select * from table1
GO
SET SHOWPLAN_TEXT OFF
GO
```

```
  |--Compute Scalar(DEFINE:([archetype_rimarqu].[dbo].[table1].[col3]=[archetype_rimarqu].[dbo].[table1].[col3
       |--Table Scan(OBJECT:([archetype_rimarqu].[dbo].[table1]))
```

Note that, even if persisted, it means that the value will change even if one of the values used for the computation changes, for example we have the following values for the first row:

```
select * from table1 where col1 = 1
```

```
col1        col2        col3
----------- ----------- -----------
1           2           3
```

Now let´s change the value for col2 to 9 - this means that col3 as to change to 10 since 1 + 9 = 10:

```
update table1 set col2 = 9 where col1 = 1
go
select * from table1 where col1 = 1
go
```

```
col1        col2        col3
----------- ----------- -----------
1           9           10
```

When speaking about reads, persisted computed columns will have an advantage when compared with non persisted ones since it will not require a computation of the value on each select for each row, but is penalized on terms of writes (since the value for each row needs to be calculated for each row inserted).

Part if the execution plan XML when inserting one row on the table used in this demos with a non persisted computed column:

```
<Object Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" IndexKind="Heap" Storag
<SetPredicate>
  <ScalarOperator ScalarString="[archetype_rimarqu].[dbo].[table1].[col1] = [@1],[archetype_ri
    <ScalarExpressionList>
      <ScalarOperator>
        <MultipleAssign>
          <Assign>
            <ColumnReference Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" Co
            <ScalarOperator>
              <Identifier>
                <ColumnReference Column="@1" />
              </Identifier>
            </ScalarOperator>
          </Assign>
          <Assign>
            <ColumnReference Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" Co
            <ScalarOperator>
              <Identifier>
                <ColumnReference Column="@2" />
              </Identifier>
            </ScalarOperator>
```

With a persisted computed column:

```
<ScalarInsert DMLRequestSort="false">
                <DefinedValues>
                  <DefinedValue>
                    <ColumnReference Column="Expr1003" />
                    <ScalarOperator ScalarString="[@1]+[@2]">
                      <Arithmetic Operation="ADD">
                        <ScalarOperator>
                          <Identifier>
                            <ColumnReference Column="@1" />
                          </Identifier>
                        </ScalarOperator>
                        <ScalarOperator>
                          <Identifier>
                            <ColumnReference Column="@2" />
                          </Identifier>
                        </ScalarOperator>
                      </Arithmetic>
                    </ScalarOperator>
                  </DefinedValue>
                </DefinedValues>
                <Object Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" IndexKind="Heap" Storag
                <SetPredicate>
                  <ScalarOperator ScalarString="[archetype_rimarqu].[dbo].[table1].[col1] = [@1],[archetype_ri
                    <ScalarExpressionList>
                      <ScalarOperator>
                        <MultipleAssign>
                          <Assign>
                            <ColumnReference Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" Co
                            <ScalarOperator>
                              <Identifier>
                                <ColumnReference Column="@1" />
                              </Identifier>
                            </ScalarOperator>
                          </Assign>
                          <Assign>
                            <ColumnReference Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" Co
                            <ScalarOperator>
                              <Identifier>
                                <ColumnReference Column="@2" />
                              </Identifier>
                            </ScalarOperator>
                          </Assign>
                          <Assign>
                            <ColumnReference Database="[archetype_rimarqu]" Schema="[dbo]" Table="[table1]" Co
                            <ScalarOperator>
                              <Identifier>
                                <ColumnReference Column="Expr1003" />
```

To summarize, on each row inserted or updated there will be a sight overhead.

Some examples of use cases

- Customer comes from Oracle, where he uses function-based indexes. A persisted computed column with an index would be the closest feature on SQL Server when compared with this type of indexes.
- Customer wants to partition a table by day. He has a datetime column (contains the date and hour) but he wants to simplify the partition column value by only day. This can be achieved with a persisted computed column. For example:

```
-- One value for each day of the month - 31
CREATE PARTITION FUNCTION myRangePF1 (int)
    AS RANGE LEFT FOR VALUES (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30

-- partition scheme for each value + for any value that doesnt fall on the condition (32 partitions)
CREATE PARTITION SCHEME myRangePS1
    AS PARTITION myRangePF1
    TO ([PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[P
        [PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[PRIMARY],[P
GO


CREATE TABLE "dbo"."part_table" (
    "id"    bigint IDENTITY(1,1) NOT NULL,
       PartitionCol AS CAST(DATEPART(day, datecol) AS INT)
PERSISTED NOT NULL, -- partition column
    "datecol"  datetime NOT NULL,
    "col1"  int NOT NULL)
       ON myRangePS1 (PartitionCol) -- specify the Partition function and the column used by partitioning
GO
alter table "dbo"."part_table" add primary key clustered (id,PartitionCol)

-- insert some rows

declare @i int = 0;


while @i < 32
begin
insert into part_table (datecol,col1) values ((select getdate()-@i),@i)

set @i+=1
end
```

Public Doc Reference

Specify Computed Columns in a Table

More information

An overview of computed columns in SQL Server

How good have you found this content?