(T34)討論 Over、RowNumber。比較 Rank 和 DenseRnk。討論 Running 和 TotalNTile
CourseGUID: e48417fc-9db5-4e99-822c-706c5ccef6cc
==========
(T34)討論 Over、RowNumber。比較 Rank 和 DenseRnk。討論 Running 和 TotalNTile
==========

8. Query - First_Value()
8.1. Create Sample data - There are some duplicate GameScore
8.2. FIRST_VALUE(C1) OVER (ORDER BY C2) AS AliasName
8.3. FIRST_VALUE(C1) OVER (PARTITION BY C3 ORDER BY C2) AS AliasName
8.4. Clean up
---
9. Window Functions
9.1. Create Sample data - There are some duplicate GameScore
9.2. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
9.3. AVG(GameScore) OVER (ORDER BY GameScore ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS AvgGameScore
9.4. ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS AvgGameScore
9.5. Clean up
---
10. Rows And Range
10.1. Create Sample data - There is no duplicate GameScore
10.2. When there is no duplicate GameScore, The following clauses are equivalent
10.3. Create Sample data - There are some duplicate GameScore
10.4. When there are some duplicate GameScore, Rows and Range treate duplicate differently.
10.5. Create Sample data, There are some duplicate GameScore
10.6. There are some duplicate GameScore
10.6.1. ORDER BY GameScore
10.6.2. PARTITION BY Gender ORDER BY GameScore
10.6.3. Logic Error - (ORDER BY GameScore) With (PARTITION BY Gender ORDER BY GameScore)
---
11. Last Value Function
11.1. Create Sample data - There are some duplicate GameScore
11.2. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW : The following clauses are equivalent
11.3. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
11.4. ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
11.5. PARTITION BY C2 ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
11.6. PARTITION BY C2 ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
---
12. Find Nth highest GameScore
12.1. Create Sample Data
12.2. Get the highest GameScore
12.3. Get the 2nd highest GameScore
12.4. Get the Nth highest GameScore by subQuery
12.5. Revise RANK() OVER (ORDER BY C1, C2, ...) / DENSE_RANK() OVER (ORDER BY C1, C2, ...) / ROW_NUMBER() OVER (ORDER BY C1, C2, ...)
12.6. Get the Nth highest GameScore by CTE and DENSE_RANK()
12.7. Get the Nth highest GameScore by CTE and ROW_NUMBER()
12.8. Clean up
==========

0. Summary

1.
Over clause Syntax
--SELECT
-- ... non-aggregated columns ...
-- aggregatedFunction(C1) OVER (PARTITION BY C1,C2,C3...) AS AliasName ,
--FROM   TableName;
1.1.
I cannot SELECT non-aggregated columns in the GROUP BY query.
If I want to do so, I can use INNER JOIN , or
function (...) OVER (PARTITION BY C1, C2, ...)
1.2.
OVER (PARTITION BY C1,C2,C3...) means ORDER BY C1,C2,C3....
Then SELECT aggregatedFunction(C1)   AS AliasName.
aggregatedFunction can be Count, Sum, Avg, Min, Max
1.3.
The following clauses are equivalent:
1.3.1.
E.g.1.
```
--SELECT  Name ,
--       p.Salary ,
--       p.Gender ,
--       GenderGroup.AvgSalary ,
--       GenderGroup.MinSalary ,
--       GenderGroup.MaxSalary
--FROM   PersonA p
--INNER JOIN ( SELECT Gender ,
--                    AVG(Salary) AS AvgSalary ,
--                    MIN(Salary) AS MinSalary ,
--                    MAX(Salary) AS MaxSalary
--             FROM   PersonA
--             GROUP BY Gender
--           ) AS GenderGroup
--ON     p.Gender = GenderGroup.Gender;
```
1.3.2.
E.g.2.
```
--SELECT  Name ,    --non-aggregated columns
--       Salary ,  --non-aggregated columns
--       Gender ,  --non-aggregated columns
--       AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
--       MIN(Salary) OVER ( PARTITION BY Gender ) AS MinSalary ,
--       MAX(Salary) OVER ( PARTITION BY Gender ) AS MaxSalary
--FROM   PersonA;
```
1.3.2.1.
```
--SELECT
--  ... non-aggregated columns  ...
--  AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
--FROM   PersonA;
```
OVER (PARTITION BY Gender) means ORDER BY Gender.
Then SELECT AVG(Salary)   AS AvgSalary

2.
```
--ROW_NUMBER() Function
```

Syntax:
--ROW_NUMBER() OVER (PARTITION BY C1 ORDER BY C1) AS AliasName
2.1.
categorise the data rows by C1 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by C1
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
2.2.
--ROW_NUMBER() OVER (ORDER BY C1) AS AliasName
Returns the sequential row number and it starts from 1.
2.3.
ORDER BY is compulsory, and PARTITION BY is optional.
If using PARTITION BY,
then row number is reset to 1 when the partition changes.
2.4.
2.4.1.
E.g.1.
--ROW_NUMBER() OVER (ORDER BY Gender) AS RowNumber1,
Order the row by Gender and give a row number which starts from 1.
2.4.2.
--ROW_NUMBER() OVER (PARTITION BY Gender ORDER BY Gender) AS RowNumber2
categorise the data rows by Gender to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Gender
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.

3.
RANK() and DENSE_RANK()
Syntax :
--RANK() OVER (ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (ORDER BY C1, C2, ...)
--RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
3.1.
--RANK() OVER (ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (ORDER BY C1, C2, ...)
Both RANK() and DENSE_RANK() returns
the sequential Rank number by C1 and it starts from 1.
Rank function skips ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
3.2.
--RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
3.2.1.
Both RANK() and DENSE_RANK() categorise
the data rows by C1 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the Rank by C1, C2, ...
and give a Rank number which starts from 1.

The Rank number is reset to 1 when the PARTITION changes.
3.2.2.
Rank function skips ranking(s) if there is a tie (平局)
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
3.3.
ORDER BY is compulsory, and PARTITION BY is optional.
If using PARTITION BY,
then Rank number is reset to 1 when the partition changes.
3.4.
3.4.1.
E.g.
--RANK() OVER (ORDER BY GameScore DESC) AS [Rank]
--DENSE_RANK() OVER (ORDER BY GameScore DESC) AS DenseRank
Both RANK() and DENSE_RANK() returns
the sequential Rank number by GameScore and it starts from 1.
Rank function skips ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
3.4.2.
E.g.
--RANK() OVER (PARTITION BY Gender ORDER BY GameScore DESC) AS [Rank]
--DENSE_RANK() OVER (PARTITION BY Gender ORDER BY GameScore DESC) AS DenseRank
3.4.2.1.
Both RANK() and DENSE_RANK() categorise
the data rows by GameScore to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the Rank by GameScore
and give a Rank number which starts from 1.
The Rank number is reset to 1 when the PARTITION changes.
3.4.2.2.
Rank function skips ranking(s) if there is a tie (平局)
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
3.5.
When to use RANK() or DENSE_RANK() ?
3.5.1.
scenario01:
The online game display the rank.
In this case,
we can use both RANK() or DENSE_RANK()
3.5.2.
scenario02:
The game competition hoster only offers reward prizes to top 3 Gamer.
The hoster can not offer anything to the top 4th Gamer.
In this case,
we use RANK() which returns 1, 1, 3, 4, 5.
The reward prizes can only give to 1,1,3.

4.
Compare ROW_NUMBER(), RANK(), and DENSE_RANK()
4.1.
If there is no duplicate data row,
there is no different in ROW_NUMBER(), RANK(), or DENSE_RANK()
4.2.
If there are some duplicate data rows,
All ROW_NUMBER(), RANK(), and DENSE_RANK() return
an increasing unique number for each row starting at 1.
4.2.1.
ROW_NUMBER() still returns different Row Number
if it meets duplicate data rows.
E.g. 1,2,3,4,5,6,7,8,9,10
4.2.2.
Both RANK() and DENSE_RANK() return the same Rank Number
if it meets duplicate data rows.
However,
Rank function skips ranking(s) if there is a tie (平局).
E.g. 1,1,1,4,5,5,7,8,9,9
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. 1,1,1,2,3,3,4,5,6,6

5.
Running Total
5.1.
--SUM(C1) OVER (ORDER BY C2) AS RunningTotal
Compute running total of C1 ORDER BY C2 without partitions
OVER (ORDER BY C2) means ORDER BY C2.
Then SELECT SUM(C1) AS RunningTotal
This will compute running total of C1 without partitions.
5.1.1.
E.g.
--SUM(GameScore) OVER (ORDER BY Id) AS RunningTotal
Compute running total of GameScore ORDER BY Id without partitions
OVER (ORDER BY Id) means ORDER BY Id.
Then SELECT SUM(GameScore) AS RunningTotal
This will compute running total of GameScore without partitions.
E.g. 1500, 1500+2600=4100, 1500+2600+3500=7600, ...etc.

5.2.
--SUM(C1) OVER (PARTITION BY C2 ORDER BY C3) AS RunningTotal
Compute running total of C1 ORDER BY C3 with partitions C2
categorise the data rows by C2 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by C3
and give a SUM(C1) which
will compute running total with partitions
The SUM(C1) is reset when the partition changes.
5.2.1.
E.g.
--SUM(GameScore) OVER (PARTITION BY Gender ORDER BY Id) AS RunningTotal
Compute running total of GameScore ORDER BY Id with partitions Gender
categorise the data rows by Gender to different categories,

and put each categories into different PARTITION.
In each PARTITION, Order the row by Id
and give a SUM(GameScore) which
will compute running total with partitions
The SUM(GameScore) is reset when the partition changes.
E.g.
Female : 1500, 1500+3350=4850, 1500+3350+3350=8200, 1500+3350+3350+3500=11700
Male : 1500, 1500+2600=4100, 1500+2600+3500=7600 ...etc.

5.3.
--SUM(C1) OVER (ORDER BY C1) AS RunningTotal
Compute running total of C1 ORDER BY C1 without partitions
OVER (ORDER BY C1) means ORDER BY C1.
Then SELECT SUM(C1) AS RunningTotal
This will compute running total of C1 without partitions.
If there are some duplicate C1,
all the duplicate values will be added to the running total at once.
5.3.1.
E.g.
--SUM(GameScore) OVER (ORDER BY GameScore) AS RunningTotal
Compute running total of GameScore ORDER BY GameScore without partitions
OVER (ORDER BY GameScore) means ORDER BY GameScore.
Then SELECT SUM(GameScore) AS RunningTotal
This will compute running total of GameScore without partitions.
If there are some duplicate GameScore,
all the duplicate values will be added to the running total at once.
E.g.
1500+1500=3000, 1500+1500=3000, 1500+1500+2500=5500,
1500+1500+2500+2600=8100,
1500+1500+2500+2600+3350+3350=14800,
1500+1500+2500+2600+3350+3350=14800, ...etc.

6.
Ntile Syntax:
--NTILE (NumberOfGroups) OVER (ORDER BY C1,C2 ...) AS AliasName
--NTILE (NumberOfGroups) OVER (PARTITION BY C1 ORDER BY C1,C2 ...) AS AliasName
6.1.
Divides the rows into a specified NumberOfGroups.
6.2.
If the NumberOfGroups is not divisible,
then the groups will have different sizes.
Larger size groups always come before smaller groups.
E.g.
--NTILE (3) OVER (ORDER BY C1) AS AliasName
NTile function without PARTITION BY
divides 10 rows into 3 Groups .
Group1 size is 4, The size of Group2 and Group3 are 3.
E.g.
--NTILE (2) OVER (ORDER BY C1) AS AliasName
divides 10 rows rows into 2 Groups.
Size of each group is 5
6.3.
NTILE function will try to create as many groups as possible.

If there are 10 rows in the table.
E.g.
--NTILE (11) OVER (ORDER BY C1) AS AliasName
CAN NOT divides 10 rows rows into 11 Groups.
Hense, NTILE (11) divides 10 rows rows into 10 Groups.
6.4.
ORDER BY Clause is compulsory,
PARTITION BY clause is optional
6.5.
--NTILE (NumberOfGroups) OVER (PARTITION BY C1 ORDER BY C2) AS AliasName
NTile function with PARTITION BY :
When the data is partitioned by C1
and then ORDER BY C2,
NTile function creates the NumberOfGroups in each partition.
E.g.
--NTILE(3) OVER (PARTITION BY Gender ORDER BY GameScore) AS [Ntile]
When the data is partitioned by GENDER,
and then ORDER BY GameScore,
NTile function creates 3 groups in each partition.

7.
Lead(), Lag() Syntax :
--LEAD(C1, OffsetNumber, DefaultValue) OVER (ORDER BY C1) AS AliasName1
--LAG(C1, OffsetNumber, DefaultValue) OVER (ORDER BY C1) AS AliasName2
--LEAD(C1, OffsetNumber, DefaultValue) OVER (PARTITION BY C2 ORDER BY C1) AS AliasName3
--LAG(C1, OffsetNumber, DefaultValue) OVER (PARTITION BY C2 ORDER BY C1) AS AliasName4
7.1.
ORDER BY C1 is compulsory, PARTITION BY is optional.

7.2.
--LEAD(C1, OffsetNumber, DefaultValue) OVER (ORDER BY C1) AS AliasName1
ORDER BY C1,
LEAD(C1, OffsetNumber, DefaultValue) let you move forward (OffsetNumber) rows.
7.2.1.
That means the value of (currentRow) of LEAD(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow + OffsetNumber) row of C1.
7.2.2.
For the value of last C1 row,
the value of (CurrentRow + OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
7.2.3.
E.g.
--LEAD(C1, OffsetNumber, DefaultValue) OVER (ORDER BY C1) AS AliasName1
--LEAD(GameScore, 2, -1) OVER (ORDER BY GameScore) AS AliasName1
LEAD(GameScore, 2, -1) let you move forward (2) rows.
That means the value of (currentRow) of LEAD(GameScore, 2, -1)
will be the value of (CurrentRow + 2) row of GameScore.
For the value of last GameScore row,
the value of (CurrentRow + 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.

7.3.
--LAG(C1, OffsetNumber, DefaultValue) OVER (ORDER BY C1) AS AliasName2
ORDER BY C1,
LEAD(C1, OffsetNumber, DefaultValue) let you move back forward (OffsetNumber) rows.
7.3.1.
That means the value of (currentRow) of LAG(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow - OffsetNumber) row of C1.
7.3.2.
For the value of First C1 row,
the value of (CurrentRow - OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
7.3.3.
E.g.
--LAG(C1, OffsetNumber, DefaultValue) OVER (ORDER BY C1) AS AliasName2
--LAG(GameScore, 2, -1) OVER (ORDER BY GameScore) AS AliasName2
LAG(GameScore, 2, -1) let you move backforward (2) rows.
That means the value of (currentRow) of LAG(GameScore, 2, -1)
will be the value of (CurrentRow - 2) row of GameScore.
For the value of 1st GameScore row,
the value of (CurrentRow - 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.

7.4.
--LEAD(C1, OffsetNumber, DefaultValue) OVER (PARTITION BY C2 ORDER BY C1) AS AliasName3
PARTITION By C2, and then ORDER BY C1,
LEAD(C1, OffsetNumber, DefaultValue) let you
move forward (OffsetNumber) rows in each PARTITION.
7.4.1.
That means in each PARTITION,
the value of (currentRow) of LEAD(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow + OffsetNumber) row of C1.
7.4.2.
For the value of last C1 row,
the value of (CurrentRow + OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
7.4.3.
E.g.
--LEAD(C1, OffsetNumber, DefaultValue) OVER (PARTITION BY C2 ORDER BY C1) AS AliasName3
--LEAD(GameScore, 2, -1) OVER (PARTITION BY Gender ORDER BY GameScore) AS AliasName3
PARTITION By Gender, and then ORDER BY GameScore,
LEAD(GameScore, 2, -1) let you
move forward (2) rows in each PARTITION.
That means in each PARTITION,
the value of (currentRow) of LEAD(GameScore, 2, -1)
will be the value of (CurrentRow + 2) row of GameScore.
For the value of last GameScore row,
the value of (CurrentRow + 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if DefaultValue is specified.

7.5.
--LAG(C1, OffsetNumber, DefaultValue) OVER (PARTITION BY C2 ORDER BY C1) AS AliasName4
PARTITION By C2, and then ORDER BY C1,
LAG(C1, OffsetNumber, DefaultValue) let you
move backforward (OffsetNumber) rows in each PARTITION.
7.5.1.
That means in each PARTITION,
the value of (currentRow) of LEAD(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow - OffsetNumber) row of C1.
7.5.2.
For the value of first C1 row,
the value of (CurrentRow - OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
7.5.3.
--LAG(C1, OffsetNumber, DefaultValue) OVER (PARTITION BY C2 ORDER BY C1) AS AliasName4
--LAG(GameScore, 1, -1) OVER (PARTITION BY Gender ORDER BY GameScore) AS AliasName4
PARTITION By Gender, and then ORDER BY GameScore,
LAG(GameScore, 1, -1) let you
move backforward (1) rows in each PARTITION.
That means in each PARTITION,
the value of (currentRow) of LEAD(GameScore, 1, -1)
will be the value of (CurrentRow - 1) row of GameScore.
For the value of first GameScore row,
the value of (CurrentRow - 1) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.

8.
First_Value() Syntax:
--FIRST_VALUE(C1) OVER (ORDER BY C2) AS AliasName
--FIRST_VALUE(C1) OVER (PARTITION BY C3 ORDER BY C2) AS AliasName
8.1.
ORDER BY C1 is compulsory, PARTITION BY is optional.
It returns the first value from the specified column

8.2.
E.g.
--FIRST_VALUE(C1) OVER (ORDER BY C2) AS AliasName
--FIRST_VALUE([Name]) OVER (ORDER BY GameScore DESC) AS No1Gamer
FIRST_VALUE() returns the name of the No1Gamer
with highest GameScore from the entire table.

8.3.
E.g.
--FIRST_VALUE(C1) OVER (PARTITION BY C3 ORDER BY C2) AS AliasName
--FIRST_VALUE([Name]) OVER (PARTITION BY Gender ORDER BY GameScore DESC) AS No1Gamer
FIRST_VALUE() returns the name of the No1Gamer
with highest GameScore from each PARTITION.

9.
Last_Value() Syntax:
--LAST_VALUE(C1) OVER (ORDER BY C2) AS AliasName

--LAST_VALUE(C1) OVER (PARTITION BY C3 ORDER BY C2) AS AliasName
8.1.
ORDER BY C1 is compulsory, PARTITION BY is optional.
It returns the last value from the specified column

8.2.
E.g.
--LAST_VALUE(C1) OVER (ORDER BY C2) AS AliasName
--LAST_VALUE([Name]) OVER (ORDER BY GameScore) AS No1Gamer
LAST_VALUE() returns the name of the No1Gamer
with highest GameScore from the entire table.

8.3.
E.g.
--LAST_VALUE(C1) OVER (PARTITION BY C3 ORDER BY C2) AS AliasName
--LAST_VALUE([Name]) OVER (PARTITION BY Gender ORDER BY GameScore) AS No1Gamer
LAST_VALUE() returns the name of the No1Gamer
with highest GameScore from each PARTITION.

10.
Rows V.S. Range
10.1.
Syntax:
10.1.1.
--SUM(C1) OVER (PARTITION BY C2 ORDER BY C1
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Range]
categorise data into different PARTITION by C2, then ORDER BY C1.
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
SUM(C1) will sum up the total of all previous rows until current row of C1
RANGE treats them as a single entity.
10.1.2.
--SUM(C1) OVER (PARTITION BY C2 ORDER BY C1) AS [Default]
The default scope is
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
Thus, categorise data into different PARTITION by C2, then ORDER BY C1.
SUM(C1) will sum up the total of all previous rows until current row of C1
RANGE treats them as a single entity.
10.1.3.
--SUM(C1) OVER (PARTITION BY C2 ORDER BY C1
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Rows]
categorise data into different PARTITION by C2, then ORDER BY C1.
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
SUM(C1) will sum up the total of all previous rows until current row of C1
ROWS treat duplicates as distinct values.

10.2.
Rows and Range scope:
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
Between all previous rows until current row.
--BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
Between all previous rows until all following rows.
--ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
Between previous 1 row until following 1 row

10.3.
E.g.
```
--SELECT  * ,
--       --PARTITION BY Gender ORDER BY GameScore
--       SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Default] ,
--       SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range] ,
--       SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
--FROM    Gamer;
```

10.3.1.
Output as the following
```
--Id Name   Gender GameScore Default Rang  Rows
--4  Name04 Female 1500      1500    1500  1500
--5  Name05 Female 3350      8200    8200  4850
--6  Name06 Female 3350      8200    8200  8200
--7  Name07 Female 3500      11700   11700 11700
--1  Name01 Male   1500      1500    1500  1500
--10 Name10 Male   2500      4000    4000  4000
--2  Name02 Male   2600      6600    6600  6600
--9  Name09 Male   3450      10050   10050 10050
--3  Name03 Male   3500      17050   17050 13550
--8  Name08 Male   3500      17050   17050 17050
```

10.3.2.
ROWS V.S. RANGE
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all the previous rows until current row.
ROWS and RANGE treat duplicate data differently.

10.3.2.1.
```
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
```
ROWS treat duplicates as distinct values.
Thus,
SUM(GameScore) for Female will return
1500, 1500+3350=4850, 1500+3350+3350=8200,
1500+3350+3350+3500=11700
SUM(GameScore) for Male will return
1500, 1500+2500=4000, 1500+2500+2600=6600,
1500+2500+2600+3450=10050,
1500+2500+2600+3450+3500=13550,
1500+2500+2600+3450+3500+3500=17050

10.3.2.2.
```
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
```
RANGE treats them as a single entity.
Thus,
SUM(GameScore) for Female will return
1500, 1500+3350+3350=8200, 1500+3350+3350=8200,

1500+3350+3350+3500=11700
SUM(GameScore) for Male will return
1500, 1500+2500=4000, 1500+2500+2600=6600,
1500+2500+2600+3450=10050,
1500+2500+2600+3450+3500+3500=17050,
1500+2500+2600+3450+3500+3500=17050

10.3.2.3.
--SUM(GameScore) OVER (PARTITION BY Gender ORDER BY GameScore) AS [Default]
Default setting is
--SUM(GameScore) OVER (PARTITION BY Gender ORDER BY GameScore
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Range]
RANGE treats them as a single entity.
Thus,
SUM(GameScore) for Female will return
1500, 1500+3350+3350=8200, 1500+3350+3350=8200,
1500+3350+3350+3500=11700
SUM(GameScore) for Male will return
1500, 1500+2500=4000, 1500+2500+2600=6600,
1500+2500+2600+3450=10050,
1500+2500+2600+3450+3500+3500=17050,
1500+2500+2600+3450+3500+3500=17050

==========

1. Query - Over Clause

```
--============================================================
--T034_01_Over.sql
--============================================================
```

1.1. Create sample data

```
--============================================================
--T034_01_01
--Create sample data
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'PersonA' ) )
    BEGIN
        TRUNCATE TABLE dbo.PersonA;
        DROP TABLE PersonA;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE PersonA
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(100) ,
  Gender NVARCHAR(10) ,
  Salary MONEY
);
GO -- Run the previous command and begins new batch
INSERT  INTO PersonA
```

```
VALUES  ( 'Name01', 'Male', 41000 );
INSERT  INTO PersonA
VALUES  ( 'Name02', 'Female', 45000 );
INSERT  INTO PersonA
VALUES  ( 'Name03', 'Male', 45000 );
INSERT  INTO PersonA
VALUES  ( 'Name04', 'Female', 41000 );
INSERT  INTO PersonA
VALUES  ( 'Name05', 'Female', 56000 );
INSERT  INTO PersonA
VALUES  ( 'Name06', 'Male', 56000 );
INSERT  INTO PersonA
VALUES  ( 'Name07', 'Female', 41000 );
INSERT  INTO PersonA
VALUES  ( 'Name08', 'Male', 65000 );
INSERT  INTO PersonA
VALUES  ( 'Name09', 'Male', 56000 );
INSERT  INTO PersonA
VALUES  ( 'Name10', 'Male', 65000 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    PersonA;
```

	Id	Name	Gender	Salary
1	1	Name01	Male	41000.00
2	2	Name02	Female	45000.00
3	3	Name03	Male	45000.00
4	4	Name04	Female	41000.00
5	5	Name05	Female	56000.00
6	6	Name06	Male	56000.00
7	7	Name07	Female	41000.00
8	8	Name08	Male	65000.00
9	9	Name09	Male	56000.00
10	10	Name10	Male	65000.00

1.2. Aggregated columns

```
--=========================================================
--T034_01_02
--Aggregated columns
SELECT  Gender ,
        COUNT(*) AS NumberOfPerson ,
        AVG(Salary) AS AvgSalary ,
        MIN(Salary) AS MinSalary ,
        MAX(Salary) AS MaxSalary
FROM    PersonA
GROUP BY Gender;
/*
1.
Output as following
--Gender  NumberOfPerson  AvgSalary  MinSalary  MaxSalary
--Female      4               45750.00     41000.00     56000.00
--Male     6           54666.6666    41000.00     65000.00
```

```
2.
I cannot SELECT non-aggregated columns in the GROUP BY query.
If I want to do so, I can use  INNER JOIN , or
function (...) OVER (PARTITION BY C1, C2, ...)
*/
```

	Gender	NumberOfPerson	AvgSalary	MinSalary	MaxSalary
1	Female	4	45750.00	41000.00	56000.00
2	Male	6	54666.6666	41000.00	65000.00

1.3. use INNER JOIN to SELECT non-aggregated columns in the GROUP BY query.

```
--=========================================================
--T034_01_03
--use INNER JOIN to SELECT non-aggregated columns in the GROUP BY query.
SELECT  Name ,
        p.Salary ,
        p.Gender ,
        GenderGroup.AvgSalary ,
        GenderGroup.MinSalary ,
        GenderGroup.MaxSalary
FROM    PersonA p
INNER JOIN ( SELECT Gender ,
                    AVG(Salary) AS AvgSalary ,
                    MIN(Salary) AS MinSalary ,
                    MAX(Salary) AS MaxSalary
             FROM   PersonA
             GROUP BY Gender
           ) AS GenderGroup
ON      p.Gender = GenderGroup.Gender;
/*
1.
Output as following
--Name      Salary      Gender   AvgSalary  MinSalary  MaxSalary
--Name02      45000.00      Female  45750.00     41000.00      56000.00
--Name04      41000.00      Female   45750.00     41000.00      56000.00
--Name05      56000.00      Female  45750.00     41000.00      56000.00
--Name07      41000.00      Female  45750.00     41000.00      56000.00
--Name08      65000.00      Male    54666.6666   41000.00      65000.00
--Name09      56000.00      Male    54666.6666   41000.00      65000.00
--Name10      65000.00      Male    54666.6666   41000.00      65000.00
--Name06      56000.00      Male    54666.6666   41000.00      65000.00
--Name03      45000.00      Male    54666.6666   41000.00      65000.00
--Name01      41000.00      Male    54666.6666   41000.00      65000.00
|__________________|  JoinColumn |_____________________________|
non-aggregated columns              aggregated columns
2.
I cannot SELECT non-aggregated columns in the GROUP BY query.
If I want to do so, I can use  INNER JOIN , or
function (...) OVER (PARTITION BY C1, C2, ...)
*/
```

	Name	Salary	Gender	AvgSalary	MinSalary	MaxSalary
1	Name02	45000.00	Female	45750.00	41000.00	56000.00
2	Name04	41000.00	Female	45750.00	41000.00	56000.00
3	Name05	56000.00	Female	45750.00	41000.00	56000.00
4	Name07	41000.00	Female	45750.00	41000.00	56000.00
5	Name08	65000.00	Male	54666.6666	41000.00	65000.00
6	Name09	56000.00	Male	54666.6666	41000.00	65000.00
7	Name10	65000.00	Male	54666.6666	41000.00	65000.00
8	Name06	56000.00	Male	54666.6666	41000.00	65000.00
9	Name03	45000.00	Male	54666.6666	41000.00	65000.00
10	Name01	41000.00	Male	54666.6666	41000.00	65000.00

non-aggregated columns JoinColumn aggregated columns

1.4. use function (...) OVER (PARTITION BY C1, C2, ...)

```
--=======================================================
--T034_01_04
--use function (...) OVER (PARTITION BY C1, C2, ...)
--to SELECT non-aggregated columns in the GROUP BY query.
SELECT  Name ,
        Salary ,
        Gender ,
        AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
        MIN(Salary) OVER ( PARTITION BY Gender ) AS MinSalary ,
        MAX(Salary) OVER ( PARTITION BY Gender ) AS MaxSalary
FROM    PersonA;
/*
1.
Output as following
--Name      Salary      Gender   AvgSalary  MinSalary  MaxSalary
--Name02      45000.00      Female  45750.00     41000.00      56000.00
--Name04      41000.00      Female   45750.00     41000.00      56000.00
--Name05      56000.00      Female  45750.00     41000.00      56000.00
--Name07      41000.00      Female  45750.00     41000.00      56000.00
--Name08      65000.00      Male    54666.6666   41000.00      65000.00
--Name09      56000.00      Male    54666.6666   41000.00      65000.00
--Name10      65000.00      Male    54666.6666   41000.00      65000.00
--Name06      56000.00      Male    54666.6666   41000.00      65000.00
--Name03      45000.00      Male    54666.6666   41000.00      65000.00
--Name01      41000.00      Male    54666.6666   41000.00      65000.00
|__________________|  JoinColumn |_____________________________|
non-aggregated columns              aggregated columns
2.
Over clause Syntax
--SELECT
--    ... non-aggregated columns  ...
--    aggregatedFunction(C1) OVER ( PARTITION BY C1,C2,C3... ) AS AliasName ,
--FROM    TableName;
2.1.
I cannot SELECT non-aggregated columns in the GROUP BY query.
If I want to do so, I can use  INNER JOIN , or
function (...) OVER (PARTITION BY C1, C2, ...)
2.2.
OVER ( PARTITION BY C1,C2,C3... ) means ORDER BY C1,C2,C3....
Then SELECT aggregatedFunction(C1)   AS AliasName.
aggregatedFunction can be Count, Sum, Avg, Min, Max
2.3.
The following clauses are equivalent:
```

```
2.3.1.
E.g.1.
--SELECT  Name ,
--        p.Salary ,
--        p.Gender ,
--        GenderGroup.AvgSalary ,
--        GenderGroup.MinSalary ,
--        GenderGroup.MaxSalary
--FROM    PersonA p
--INNER JOIN ( SELECT Gender ,
--                    AVG(Salary) AS AvgSalary ,
--                    MIN(Salary) AS MinSalary ,
--                    MAX(Salary) AS MaxSalary
--             FROM   PersonA
--             GROUP BY Gender
--           ) AS GenderGroup
--ON      p.Gender = GenderGroup.Gender;
2.3.2.
E.g.2.
--SELECT  Name ,    --non-aggregated columns
--        Salary ,  --non-aggregated columns
--        Gender ,  --non-aggregated columns
--        AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
--        MIN(Salary) OVER ( PARTITION BY Gender ) AS MinSalary ,
--        MAX(Salary) OVER ( PARTITION BY Gender ) AS MaxSalary
--FROM    PersonA;
2.3.2.1.
--SELECT
--   ... non-aggregated columns  ...
--   AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
--FROM    PersonA;
OVER ( PARTITION BY Gender ) means ORDER BY Gender.
Then SELECT AVG(Salary)   AS AvgSalary
*/
```

	Name	Salary	Gender	AvgSalary	MinSalary	MaxSalary
1	Name02	45000.00	Female	45750.00	41000.00	56000.00
2	Name04	41000.00	Female	45750.00	41000.00	56000.00
3	Name05	56000.00	Female	45750.00	41000.00	56000.00
4	Name07	41000.00	Female	45750.00	41000.00	56000.00
5	Name08	65000.00	Male	54666.6666	41000.00	65000.00
6	Name09	56000.00	Male	54666.6666	41000.00	65000.00
7	Name10	65000.00	Male	54666.6666	41000.00	65000.00
8	Name06	56000.00	Male	54666.6666	41000.00	65000.00
9	Name03	45000.00	Male	54666.6666	41000.00	65000.00
10	Name01	41000.00	Male	54666.6666	41000.00	65000.00

non-aggregated columns JoinColumn aggregated columns

1.5. Clean up

```
--=========================================================
--T034_01_05
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'PersonA' ) )
   BEGIN
       TRUNCATE TABLE dbo.PersonA;
       DROP TABLE PersonA;
```

```
    END;
GO -- Run the previous command and begins new batch
```

2. Row_Number Function

```
--=========================================================
--T034_02_Row_NumberFunction
--=========================================================
/*
1.
--ROW_NUMBER() Function
Syntax:
--ROW_NUMBER() OVER ( PARTITION BY C1 ORDER BY C1 ) AS AliasName
1.1.
categorise the data rows by C1 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by C1
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
1.2.
--ROW_NUMBER() OVER ( ORDER BY C1 ) AS AliasName
Returns the sequential row number and it starts from 1.
1.3.
ORDER BY is compulsory, and PARTITION BY is optional.
If using PARTITION BY,
then row number is reset to 1 when the partition changes.
1.4.
1.4..1.
E.g.1.
--ROW_NUMBER() OVER ( ORDER BY Gender ) AS RowNumber1,
Order the row by Gender and give a row number which starts from 1.
1.4.2.
--ROW_NUMBER() OVER ( PARTITION BY Gender ORDER BY Gender ) AS RowNumber2
categorise the data rows by Gender to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Gender
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
*/
```

2.1. Create sample data

```
--=========================================================
--T034_02_01
--Create sample data
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'PersonA' ) )
    BEGIN
        TRUNCATE TABLE dbo.PersonA;
        DROP TABLE PersonA;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE PersonA
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(100) ,
```

```
  Gender NVARCHAR(10) ,
  Salary MONEY
);
GO -- Run the previous command and begins new batch
INSERT  INTO PersonA
VALUES  ( 'Name01', 'Male', 41000 );
INSERT  INTO PersonA
VALUES  ( 'Name02', 'Female', 45000 );
INSERT  INTO PersonA
VALUES  ( 'Name03', 'Male', 45000 );
INSERT  INTO PersonA
VALUES  ( 'Name04', 'Female', 41000 );
INSERT  INTO PersonA
VALUES  ( 'Name05', 'Female', 56000 );
INSERT  INTO PersonA
VALUES  ( 'Name06', 'Male', 56000 );
INSERT  INTO PersonA
VALUES  ( 'Name07', 'Female', 41000 );
INSERT  INTO PersonA
VALUES  ( 'Name08', 'Male', 65000 );
INSERT  INTO PersonA
VALUES  ( 'Name09', 'Male', 56000 );
INSERT  INTO PersonA
VALUES  ( 'Name10', 'Male', 65000 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    PersonA;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	Salary
1	1	Name01	Male	41000.00
2	2	Name02	Female	45000.00
3	3	Name03	Male	45000.00
4	4	Name04	Female	41000.00
5	5	Name05	Female	56000.00
6	6	Name06	Male	56000.00
7	7	Name07	Female	41000.00
8	8	Name08	Male	65000.00
9	9	Name09	Male	56000.00
10	10	Name10	Male	65000.00

2.2. ROW_NUMBER() OVER ((PARTITION BY C1) ORDER BY C1) AS AliasName

```
--=========================================================
--T034_02_02
--ROW_NUMBER() OVER ( (PARTITION BY C1 ) ORDER BY C1 ) AS AliasName
SELECT  Name ,
        Salary ,
        Gender ,
```

```
        AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
        MIN(Salary) OVER ( PARTITION BY Gender ) AS MinSalary ,
        MAX(Salary) OVER ( PARTITION BY Gender ) AS MaxSalary ,
        ROW_NUMBER() OVER ( ORDER BY Gender ) AS RowNumber1 ,
        ROW_NUMBER() OVER ( PARTITION BY Gender ORDER BY Gender ) AS RowNumber2
FROM    PersonA;
GO -- Run the previous command and begins new batch
```

	Name	Salary	Gender	AvgSalary	MinSalary	MaxSalary	RowNumber1	RowNumber2
1	Name02	45000.00	Female	45750.00	41000.00	56000.00	1	1
2	Name04	41000.00	Female	45750.00	41000.00	56000.00	2	2
3	Name05	56000.00	Female	45750.00	41000.00	56000.00	3	3
4	Name07	41000.00	Female	45750.00	41000.00	56000.00	4	4
5	Name08	65000.00	Male	54666.6666	41000.00	65000.00	5	1
6	Name09	56000.00	Male	54666.6666	41000.00	65000.00	6	2
7	Name10	65000.00	Male	54666.6666	41000.00	65000.00	7	3
8	Name06	56000.00	Male	54666.6666	41000.00	65000.00	8	4
9	Name03	45000.00	Male	54666.6666	41000.00	65000.00	9	5
10	Name01	41000.00	Male	54666.6666	41000.00	65000.00	10	6

non-aggregated columns | aggregated columns | ORDER BY Gender | PARTITION BY Gender ORDER BY Gender

```
/*
1.
Output as following
--Name      Salary      Gender   AvgSalary  MinSalary  MaxSalary RowNumber1 RowNumber2
--Name02      45000.00      Female  45750.00     41000.00      56000.00  1          1
--Name04      41000.00      Female   45750.00     41000.00      56000.00  2          2
--Name05      56000.00      Female  45750.00     41000.00      56000.00  3          3
--Name07      41000.00      Female  45750.00     41000.00      56000.00  4          4
--Name08      65000.00      Male    54666.6666   41000.00      65000.00  5          1
--Name09      56000.00      Male    54666.6666   41000.00      65000.00  6          2
--Name10      65000.00      Male    54666.6666   41000.00      65000.00  7          3
--Name06      56000.00      Male    54666.6666   41000.00      65000.00  8          4
--Name03      45000.00      Male    54666.6666   41000.00      65000.00  9          5
--Name01      41000.00      Male    54666.6666   41000.00      65000.00  10         6
|__________________|  JoinColumn |_____________________________|           |______|
non-aggregated columns              aggregated columns          ORDER BY   PARTITION BY Gender
                                                                 Gender    ORDER BY Gender
-----------------------------------------------------------------------
2.
Over clause Syntax
--SELECT
--    ... non-aggregated columns  ...
--    aggregatedFunction(C1) OVER ( PARTITION BY C1,C2,C3... ) AS AliasName ,
--FROM    TableName;
2.1.
I cannot SELECT non-aggregated columns in the GROUP BY query.
If I want to do so, I can use  INNER JOIN , or
function (...) OVER (PARTITION BY C1, C2, ...)
2.2.
OVER ( PARTITION BY C1,C2,C3... ) means ORDER BY C1,C2,C3....
Then SELECT aggregatedFunction(C1)   AS AliasName.
aggregatedFunction can be Count, Sum, Avg, Min, Max
2.3.
The following clauses are equivalent:
2.3.1.
E.g.1.
--SELECT  Name ,
--        p.Salary ,
--        p.Gender ,
--        GenderGroup.AvgSalary ,
--        GenderGroup.MinSalary ,
--        GenderGroup.MaxSalary
--FROM    PersonA p
```

```
--INNER JOIN ( SELECT Gender ,
--                     AVG(Salary) AS AvgSalary ,
--                     MIN(Salary) AS MinSalary ,
--                     MAX(Salary) AS MaxSalary
--              FROM   PersonA
--              GROUP BY Gender
--            ) AS GenderGroup
--ON      p.Gender = GenderGroup.Gender;
2.3.2.
E.g.2.
--SELECT  Name ,    --non-aggregated columns
--        Salary ,  --non-aggregated columns
--        Gender ,  --non-aggregated columns
--        AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
--        MIN(Salary) OVER ( PARTITION BY Gender ) AS MinSalary ,
--        MAX(Salary) OVER ( PARTITION BY Gender ) AS MaxSalary
--FROM    PersonA;
2.3.2.1.
--SELECT
--   ... non-aggregated columns  ...
--   AVG(Salary) OVER ( PARTITION BY Gender ) AS AvgSalary ,
--FROM    PersonA;
OVER ( PARTITION BY Gender ) means ORDER BY Gender.
Then SELECT AVG(Salary)   AS AvgSalary
-----------------------------------------------------------------------
3.
--ROW_NUMBER() Function
Syntax:
--ROW_NUMBER() OVER ( PARTITION BY C1 ORDER BY C1 ) AS AliasName
3.1.
categorise the data rows by C1 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by C1
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
3.2.
--ROW_NUMBER() OVER ( ORDER BY C1 ) AS AliasName
Returns the sequential row number and it starts from 1.
3.3.
ORDER BY is compulsory, and PARTITION BY is optional.
If using PARTITION BY,
then row number is reset to 1 when the partition changes.
3.4.
3.4.1.
E.g.1.
--ROW_NUMBER() OVER ( ORDER BY Gender ) AS RowNumber1,
Order the row by Gender and give a row number which starts from 1.
3.4.2.
--ROW_NUMBER() OVER ( PARTITION BY Gender ORDER BY Gender ) AS RowNumber2
categorise the data rows by Gender to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Gender
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
*/
```

2.3. Delete duplicate rows

```
--=========================================================
--T034_02_03
--Delete duplicate rows
-------------------------------------------------------------------------
--T034_02_03_01
--Create Sample Data again
IF ( EXISTS ( SELECT    *
```

```sql
            FROM      INFORMATION_SCHEMA.TABLES
            WHERE     TABLE_NAME = 'PersonA' ) )
    BEGIN
        TRUNCATE TABLE dbo.PersonA;
        DROP TABLE PersonA;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE PersonA
(
  Id INT ,
  [Name] NVARCHAR(100) ,
  Gender NVARCHAR(10) ,
  Salary MONEY
);
GO -- Run the previous command and begins new batch
INSERT  INTO PersonA
VALUES  ( 1, 'Name01', 'Male', 41000 );
INSERT  INTO PersonA
VALUES  ( 2, 'Name02', 'Female', 45000 );
INSERT  INTO PersonA
VALUES  ( 3, 'Name03', 'Male', 45000 );
INSERT  INTO PersonA
VALUES  ( 1, 'Name01', 'Male', 41000 );
INSERT  INTO PersonA
VALUES  ( 2, 'Name02', 'Female', 45000 );
INSERT  INTO PersonA
VALUES  ( 3, 'Name03', 'Male', 45000 );
INSERT  INTO PersonA
VALUES  ( 1, 'Name01', 'Male', 41000 );
INSERT  INTO PersonA
VALUES  ( 2, 'Name02', 'Female', 45000 );
INSERT  INTO PersonA
VALUES  ( 3, 'Name03', 'Male', 45000 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.PersonA;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	Salary
1	1	Name01	Male	41000.00
2	2	Name02	Female	45000.00
3	3	Name03	Male	45000.00
4	1	Name01	Male	41000.00
5	2	Name02	Female	45000.00
6	3	Name03	Male	45000.00
7	1	Name01	Male	41000.00
8	2	Name02	Female	45000.00
9	3	Name03	Male	45000.00

```sql
--------------------------------------------------------------------------
--T034_02_03_02
--Delete duplicate rows
```

```
WITH    PersonACTE
          AS ( SELECT   * ,
                        ROW_NUMBER() OVER ( PARTITION BY Id ORDER BY Id ) AS RowNumber
               FROM     PersonA
             )
    DELETE  FROM PersonACTE
    WHERE   RowNumber > 1;
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.PersonA;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	Salary
1	1	Name01	Male	41000.00
2	3	Name03	Male	45000.00
3	2	Name02	Female	45000.00

```
/*
1.
--ROW_NUMBER() Function
Syntax:
--ROW_NUMBER() OVER ( PARTITION BY C1 ORDER BY C1 ) AS AliasName
1.1.
categorise the data rows by C1 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by C1
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
1.2.
--ROW_NUMBER() OVER ( ORDER BY C1 ) AS AliasName
Returns the sequential row number and it starts from 1.
1.3.
ORDER BY is compulsory, and PARTITION BY is optional.
If using PARTITION BY,
then row number is reset to 1 when the partition changes.
1.4.
1.4.1.
E.g.1.
--ROW_NUMBER() OVER ( ORDER BY Gender ) AS RowNumber1,
Order the row by Gender and give a row number which starts from 1.
1.4.2.
--ROW_NUMBER() OVER ( PARTITION BY Gender ORDER BY Gender ) AS RowNumber2
categorise the data rows by Gender to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Gender
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
-----------------------------------------------
2.
--SELECT   * ,
--     ROW_NUMBER() OVER ( PARTITION BY Id ORDER BY Id ) AS RowNumber
--FROM     PersonA
2.1.
Output as the following.
--Id  Name    Gender  Salary   RowNumber
--1      Name01  Male    41000.00  1
--1      Name01  Male    41000.00  2
--1      Name01  Male    41000.00  3
--2      Name02  Female  45000.00  1
--2      Name02  Female  45000.00  2
--2      Name02  Female  45000.00  3
--3      Name03  Male    45000.00  1
--3      Name03  Male    45000.00  2
--3      Name03  Male    45000.00  3
```

```
2.2.
--     ROW_NUMBER() OVER ( PARTITION BY Id ORDER BY Id ) AS RowNumber
categorise the data rows by Id to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Id
and give a row number which starts from 1.
The row number is reset to 1 when the partition changes.
2.3.
--DELETE  FROM PersonACTE
--WHERE   RowNumber > 1;
Previously, we separate the rows to different PARTITION.
I realised that the rows in each PARTITION are actully duplicate rows.
Thus, we spare the RowNumber=1,
and we delete  RowNumber > 1  to delete all the duplicate rows.
*/
```

2.4. Clean up

```
--=========================================================
--T034_02_04
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'PersonA' ) )
    BEGIN
        TRUNCATE TABLE dbo.PersonA;
        DROP TABLE PersonA;
    END;
GO -- Run the previous command and begins new batch
```

3. Rank(), DenseRank()

```
--=========================================================
--T034_03_Rank_DenseRank
--=========================================================
/*
1.
RANK() and DENSE_RANK()
Syntax :
--RANK() OVER (ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (ORDER BY C1, C2, ...)
--RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
1.1.
--RANK() OVER (ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (ORDER BY C1, C2, ...)
Both RANK() and DENSE_RANK() returns
the sequential Rank number by C1 and it starts from 1.
Rank function skips ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
1.2.
--RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
1.2.1.
Both RANK() and DENSE_RANK() categorise
the data rows by C1 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the Rank by C1, C2, ...
and give a Rank number which starts from 1.
```

```
The Rank number is reset to 1 when the PARTITION changes.
1.2.2.
Rank function skips ranking(s) if there is a tie (平局)
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
1.3.
ORDER BY is compulsory, and PARTITION BY is optional.
If using PARTITION BY,
then Rank number is reset to 1 when the partition changes.
1.4.
1.4.1.
E.g.
--RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank]
--DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
Both RANK() and DENSE_RANK() returns
the sequential Rank number by GameScore and it starts from 1.
Rank function skips ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
1.4.2.
E.g.
--RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS [Rank]
--DENSE_RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS DenseRank
1.4.2.1.
Both RANK() and DENSE_RANK() categorise
the data rows by GameScore to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the Rank by GameScore
and give a Rank number which starts from 1.
The Rank number is reset to 1 when the PARTITION changes.
1.4.2.2.
Rank function skips ranking(s) if there is a tie (平局)
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
1.5.
When to use RANK() or DENSE_RANK() ?
1.5.1.
scenario01:
The online game display the rank.
In this case,
we can use both RANK() or DENSE_RANK()
1.5.2.
scenario02:
The game competition hoster only offers reward prizes to top 3 Gamer.
The hoster can not offer anything to the top 4th Gamer.
In this case,
we use RANK() which returns 1, 1, 3, 4, 5.
The reward prizes can only give to 1,1,3.
*/
```

3.1. Create Sample Data

```
--============================================================
--T034_03_01
--Create Sample Data
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
   BEGIN
       TRUNCATE TABLE dbo.Gamer;
       DROP TABLE Gamer;
```

```
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

3.2. RANK() OVER (ORDER BY C1, C2, ...) V.S. DENSE_RANK() OVER (ORDER BY C1, C2, ...)

```
--============================================================
```

```
--T034_03_02
--RANK() OVER (ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (ORDER BY C1, C2, ...)
SELECT  Name ,
        GameScore ,
        Gender ,
        RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank] ,
        DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Name	GameScore	Gender	Rank	DenseRank
1	Name03	3500	Male	1	1
2	Name07	3500	Female	1	1
3	Name08	3500	Male	1	1
4	Name09	3450	Male	4	2
5	Name05	3350	Female	5	3
6	Name06	3350	Female	5	3
7	Name02	2600	Male	7	4
8	Name10	2500	Male	8	5
9	Name01	1500	Male	9	6
10	Name04	1500	Female	9	6

```
/*
1.
Output as following.
--Name   GameScore  Gender Rank DenseRank
--Name03      3500   Male   1      1
--Name07      3500   Female 1      1
--Name08      3500   Male   1      1
--Name09      3450   Male   4      2
--Name05      3350   Female 5      3
--Name06      3350   Female 5      3
--Name02      2600   Male   7      4
--Name10      2500   Male   8      5
--Name01      1500   Male   9      6
--Name04      1500   Female 9      6
2.
--RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank]
--DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
Both RANK() and DENSE_RANK() returns
the sequential Rank number by GameScore and it starts from 1.
Rank function skips ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
*/
```

3.3. RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...) V.S. DENSE_RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)

```
--============================================================
--T034_03_03
--RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
```

```
--DENSE_RANK() OVER (PARTITION BY C1 ORDER BY C1, C2, ...)
SELECT  Name ,
        GameScore ,
        Gender ,
        RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS [Rank] ,
        DENSE_RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS DenseRank
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Name	GameScore	Gender	Rank	DenseRank
1	Name07	3500	Female	1	1
2	Name05	3350	Female	2	2
3	Name06	3350	Female	2	2
4	Name04	1500	Female	4	3
5	Name03	3500	Male	1	1
6	Name08	3500	Male	1	1
7	Name09	3450	Male	3	2
8	Name02	2600	Male	4	3
9	Name10	2500	Male	5	4
10	Name01	1500	Male	6	5

```
/*
1.
Output as following.
--Name   GameScore  Gender Rank DenseRank
--Name07     3500   Female 1      1
--Name05     3350   Female 2      2
--Name06     3350   Female 2      2
--Name04     1500   Female 4      3
--Name03     3500   Male   1      1
--Name08     3500   Male   1      1
--Name09     3450   Male   3      2
--Name02     2600   Male   4      3
--Name10     2500   Male   5      4
--Name01     1500   Male   6      5
2.
--RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS [Rank]
--DENSE_RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS DenseRank
2.1.
Both RANK() and DENSE_RANK() categorise
the data rows by GameScore to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the Rank by GameScore
and give a Rank number which starts from 1.
The Rank number is reset to 1 when the PARTITION changes.
2.2.
Rank function skips ranking(s) if there is a tie (平局)
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
*/
```

3.4. RANK() and DENSE_RANK() with common table expression (CTE)

```
--============================================================
--T034_03_04
--RANK() and DENSE_RANK() with common table expression (CTE)
```

```
-------------------------------
--T034_03_04_01
WITH    GameResultCte
          AS ( SELECT   * ,
                        RANK() OVER ( ORDER BY GameScore DESC ) AS GameScoreRank
               FROM     Gamer
             )
    SELECT TOP 1
            *
    FROM    GameResultCte
    WHERE   GameScoreRank = 2;
GO -- Run the previous command and begins new batch
-------------------------------
--T034_03_04_02
WITH    GameResultCte
          AS ( SELECT   * ,
                        DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS GameScoreRank
               FROM     Gamer
             )
    SELECT TOP 1
            *
    FROM    GameResultCte
    WHERE   GameScoreRank = 2;
GO -- Run the previous command and begins new batch
-------------------------------
--T034_03_04_03
WITH    GameResultCte
          AS ( SELECT   * ,
                        DENSE_RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS GameScoreRank
               FROM     Gamer
             )
    SELECT TOP 1
            *
    FROM    GameResultCte
    WHERE   GameScoreRank = 3
            AND Gender = 'Female';
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	GameScoreRank

	Id	Name	Gender	GameScore	GameScoreRank
1	9	Name09	Male	3450	2

	Id	Name	Gender	GameScore	GameScoreRank
1	4	Name04	Female	1500	3

```
/*
1.
--SELECT  Name ,
--        GameScore ,
--        Gender ,
--        RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank] ,
--        DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
--FROM    Gamer
--WHERE   DenseRank = 2  --Syntax Error
```

```
--WHERE   [Rank] = 2     --Syntax Error
We can not use WHERE cluase dirrectly to OVER cluase.
Thus, we need to use common table expression (CTE)
2.
Rank function skips ranking(s) if there is a tie (平局)
Rank function will NOT skip ranking(s) if there is a tie (平局)
2.1.
The First Query.
--RANK() OVER ( ORDER BY GameScore DESC ) AS GameScoreRank
This will return 1,1,1,4,5,5,7,8,9,9
--WHERE   GameScoreRank = 2;
This will return nothing.
2.2.
The second Query.
--DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS GameScoreRank
This will return 1,1,1,2,3,3,4,5,6,6
2.3.
--DENSE_RANK() OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS GameScoreRank
This will return
Female : 1,2,2,3
Male : 1,1,2,3,4,5
--WHERE   GameScoreRank = 3
--     AND Gender = 'Female';
This will return
Female : 3
*/
```

3.5. Clean up

```
--============================================================
--Ch110_05
--Clean up
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gammer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gammer;
        DROP TABLE Gammer;
    END;
GO -- Run the previous command and begins new batch
```

4. Compare ROW_NUMBER(), RANK(), and DENSE_RANK()

```
--============================================================
--T034_04_Compare_RowNumber_Rank_DenseRank
--============================================================
/*
1.
Compare ROW_NUMBER(), RANK(), and DENSE_RANK()
1.1.
If there is no duplicate data row,
there is no different in ROW_NUMBER(), RANK(), or DENSE_RANK()
1.2.
If there are some duplicate data rows,
All ROW_NUMBER(), RANK(), and DENSE_RANK() return
an increasing unique number for each row starting at 1.
1.2.1.
ROW_NUMBER() still returns different Row Number
if it meets duplicate data rows.
```

```
E.g. 1,2,3,4,5,6,7,8,9,10
1.2.2.
Both RANK() and DENSE_RANK() return the same Rank Number
if it meets duplicate data rows.
However,
Rank function skips ranking(s) if there is a tie (平局).
E.g. 1,1,1,4,5,5,7,8,9,9
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. 1,1,1,2,3,3,4,5,6,6
*/
```

4.1. Create Sample Data - There is no duplicate GameScore

```
--============================================================
--T034_04_01
--Create Sample Data
--There is no duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1200 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 1300 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 1400 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 1600 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 1800 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 1700 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 1900 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 2000 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2100 );
GO -- Run the previous command and begins new batch
```

```
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1200
2	2	Name02	Male	1300
3	3	Name03	Male	1400
4	4	Name04	Female	1500
5	5	Name05	Female	1600
6	6	Name06	Female	1800
7	7	Name07	Female	1700
8	8	Name08	Male	1900
9	9	Name09	Male	2000
10	10	Name10	Male	2100

4.2. Compare ROW_NUMBER(), RANK(), or DENSE_RANK() - There is no duplicate GameScore

```
--============================================================
--T034_04_02
--Compare ROW_NUMBER(), RANK(), or DENSE_RANK()
--There is no duplicate GameScore
SELECT  * ,
        ROW_NUMBER() OVER ( ORDER BY GameScore DESC ) AS RowNumber ,
        RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank] ,
        DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
FROM    Gamer;
GO -- Run the previous command and begins new batch
/*
1.
Output as the following
--Id  Name   Gender  GameScore RowNumber Rank DenseRank
--10  Name10  Male   2100      1         1    1
--9   Name09  Male   2000      2         2    2
--8   Name08  Male   1900      3         3    3
--6   Name06  Female 1800      4         4    4
--7   Name07  Female 1700      5         5    5
--5   Name05  Female 1600      6         6    6
--4   Name04  Female 1500      7         7    7
--3   Name03  Male   1400      8         8    8
--2   Name02  Male   1300      9         9    9
--1   Name01  Male   1200      10        10   10
There is no duplicate GameScore,
Thus, there is no different in ROW_NUMBER(), RANK(), or DENSE_RANK()
*/
```

	Id	Name	Gender	GameScore	RowNumber	Rank	DenseRank
1	10	Name10	Male	2100	1	1	1
2	9	Name09	Male	2000	2	2	2
3	8	Name08	Male	1900	3	3	3
4	6	Name06	Female	1800	4	4	4
5	7	Name07	Female	1700	5	5	5
6	5	Name05	Female	1600	6	6	6
7	4	Name04	Female	1500	7	7	7
8	3	Name03	Male	1400	8	8	8
9	2	Name02	Male	1300	9	9	9
10	1	Name01	Male	1200	10	10	10

4.3. Create Sample Data - There are some duplicate GameScore

```
--============================================================
--T034_04_03
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
```

```
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

4.4. Compare ROW_NUMBER(), RANK(), or DENSE_RANK() - There are some duplicate GameScore

```
--============================================================
--T034_04_04
--Compare ROW_NUMBER(), RANK(), or DENSE_RANK()
--There are some duplicate GameScore
SELECT  * ,
        ROW_NUMBER() OVER ( ORDER BY GameScore DESC ) AS RowNumber ,
        RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank] ,
        DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	RowNumber	Rank	DenseRank
1	3	Name03	Male	3500	1	1	1
2	7	Name07	Female	3500	2	1	1
3	8	Name08	Male	3500	3	1	1
4	9	Name09	Male	3450	4	4	2
5	5	Name05	Female	3350	5	5	3
6	6	Name06	Female	3350	6	5	3
7	2	Name02	Male	2600	7	7	4
8	10	Name10	Male	2500	8	8	5
9	1	Name01	Male	1500	9	9	6
10	4	Name04	Female	1500	10	9	6

```
/*
1.
Output as the following
```

```
--Id  Name   Gender  GameScore RowNumber Rank DenseRank
--3   Name03 Male    3500      1         1    1
--7   Name07 Female  3500      2         1    1
--8   Name08 Male    3500      3         1    1
--9   Name09 Male    3450      4         4    2
--5   Name05 Female  3350      5         5    3
--6   Name06 Female  3350      6         5    3
--2   Name02 Male    2600      7         7    4
--10  Name10 Male    2500      8         8    5
--1   Name01 Male    1500      9         9    6
--4   Name04 Female  1500      10        9    6
2.
Compare ROW_NUMBER(), RANK(), and DENSE_RANK()
2.1.
If there is no duplicate data row,
there is no different in ROW_NUMBER(), RANK(), or DENSE_RANK()
2.2.
If there are some duplicate data rows,
All ROW_NUMBER(), RANK(), and DENSE_RANK() return
an increasing unique number for each row starting at 1.
2.2.1.
ROW_NUMBER() still returns different Row Number
if it meets duplicate data rows.
E.g. 1,2,3,4,5,6,7,8,9,10
2.2.2.
Both RANK() and DENSE_RANK() return the same Rank Number
if it meets duplicate data rows.
However,
Rank function skips ranking(s) if there is a tie (平局).
E.g. 1,1,1,4,5,5,7,8,9,9
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. 1,1,1,2,3,3,4,5,6,6
*/
```

4.5. Clean up

```
--============================================================
--T034_04_05
--Clean up
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch

======================================================================
```

5. Running Total

```
--============================================================
--T034_05_RunningTotal
--============================================================
/*
1.1.
--SUM(C1) OVER ( ORDER BY C2 ) AS RunningTotal
Compute running total of C1 ORDER BY C2 without partitions
OVER ( ORDER BY C2 ) means ORDER BY C2.
Then SELECT SUM(C1)   AS RunningTotal
This will compute running total of C1 without partitions.
```

```
1.1.1.
E.g.
--SUM(GameScore) OVER ( ORDER BY Id ) AS RunningTotal
Compute running total of GameScore ORDER BY Id without partitions
OVER ( ORDER BY Id ) means ORDER BY Id.
Then SELECT SUM(GameScore)   AS RunningTotal
This will compute running total of GameScore without partitions.
E.g. 1500, 1500+2600=4100, 1500+2600+3500=7600, ...etc.
------------------------
1.2.
--SUM(C1) OVER ( PARTITION BY C2 ORDER BY C3 ) AS RunningTotal
Compute running total of C1 ORDER BY C3 with partitions C2
categorise the data rows by C2 to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by C3
and give a SUM(C1) which
will compute running total with partitions
The SUM(C1) is reset when the partition changes.
1.2.1.
E.g.
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY Id ) AS RunningTotal
Compute running total of GameScore ORDER BY Id with partitions Gender
categorise the data rows by Gender to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Id
and give a SUM(GameScore) which
will compute running total with partitions
The SUM(GameScore) is reset when the partition changes.
E.g.
Female : 1500, 1500+3350=4850, 1500+3350+3350=8200, 1500+3350+3350+3500=11700
Male : 1500, 1500+2600=4100, 1500+2600+3500=7600 ...etc.
------------------------
1.3.
--SUM(C1) OVER ( ORDER BY C1 ) AS RunningTotal
Compute running total of C1 ORDER BY C1 without partitions
OVER ( ORDER BY C1 ) means ORDER BY C1.
Then SELECT SUM(C1)   AS RunningTotal
This will compute running total of C1 without partitions.
If there are some duplicate C1,
all the duplicate values will be added to the running total at once.
1.3.3.
E.g.
--SUM(GameScore) OVER ( ORDER BY GameScore ) AS RunningTotal
Compute running total of GameScore ORDER BY GameScore without partitions
OVER ( ORDER BY GameScore ) means ORDER BY GameScore.
Then SELECT SUM(GameScore)   AS RunningTotal
This will compute running total of GameScore without partitions.
If there are some duplicate GameScore,
all the duplicate values will be added to the running total at once.
E.g.
1500+1500=3000, 1500+1500=3000, 1500+1500+2500=5500,
1500+1500+2500+2600=8100,
1500+1500+2500+2600+3350+3350=14800,
1500+1500+2500+2600+3350+3350=14800, ...etc.
*/
```

5.1. Create Sample data - There are some duplicate GameScore

```
--============================================================
--T034_05_01
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
```

```
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
        PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

5.2. compute running total of GameScore ORDER BY Id without partitions

```
--============================================================
--T034_05_02
--compute running total of GameScore ORDER BY Id without partitions
SELECT  * ,
        SUM(GameScore) OVER ( ORDER BY Id ) AS RunningTotal
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	RunningTotal
1	1	Name01	Male	1500	1500
2	2	Name02	Male	2600	4100
3	3	Name03	Male	3500	7600
4	4	Name04	Female	1500	9100
5	5	Name05	Female	3350	12450
6	6	Name06	Female	3350	15800
7	7	Name07	Female	3500	19300
8	8	Name08	Male	3500	22800
9	9	Name09	Male	3450	26250
10	10	Name10	Male	2500	28750

```
/*
1.
Output as the following
--Id Name   Gender GameScore RunningTotal
--1  Name01 Male   1500      1500
--2  Name02 Male   2600      4100
--3  Name03 Male   3500      7600
--4  Name04 Female 1500      9100
--5  Name05 Female 3350      12450
--6  Name06 Female 3350      15800
--7  Name07 Female 3500      19300
--8  Name08 Male   3500      22800
--9  Name09 Male   3450      26250
--10 Name10 Male   2500      28750
2.
--SUM(GameScore) OVER ( ORDER BY Id ) AS RunningTotal
OVER ( ORDER BY Id ) means ORDER BY Id.
Then SELECT SUM(GameScore)   AS RunningTotal
This will compute running total of GameScore without partitions.
E.g. 1500, 1500+2600=4100, 1500+2600+3500=7600, ...etc.
*/
```

5.3. compute running total of GameScore ORDER BY Id with partitions Gender

```
--============================================================
--T034_05_03
--compute running total of GameScore ORDER BY Id with partitions Gender
SELECT  * ,
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY Id ) AS RunningTotal
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	RunningTotal
1	4	Name04	Female	1500	1500
2	5	Name05	Female	3350	4850
3	6	Name06	Female	3350	8200
4	7	Name07	Female	3500	11700
5	1	Name01	Male	1500	1500
6	2	Name02	Male	2600	4100
7	3	Name03	Male	3500	7600
8	8	Name08	Male	3500	11100
9	9	Name09	Male	3450	14550
10	10	Name10	Male	2500	17050

```
/*
1.
Output as the following
--Id Name   Gender GameScore RunningTotal ___
--4  Name04 Female 1500      1500           |
--5  Name05 Female 3350      4850           | Female
--6  Name06 Female 3350      8200           | RunningTotal
--7  Name07 Female 3500      11700        __|
--1  Name01 Male   1500      1500           |
--2  Name02 Male   2600      4100           |
--3  Name03 Male   3500      7600           | Male
--8  Name08 Male   3500      11100          | RunningTotal
--9  Name09 Male   3450      14550          |
--10 Name10 Male   2500      17050        __|
2.
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY Id ) AS RunningTotal
categorise the data rows by Gender to different categories,
and put each categories into different PARTITION.
In each PARTITION, Order the row by Id
and give a SUM(GameScore) which
will compute running total with partitions
The SUM(GameScore) is reset when the partition changes.
E.g.
Female : 1500, 1500+3350=4850, 1500+3350+3350=8200, 1500+3350+3350+3500=11700
Male : 1500, 1500+2600=4100, 1500+2600+3500=7600 ...etc.
*/
```

5.4. compute running total of GameScore ORDER BY GameScore without partitions

```
--============================================================
--T034_05_04
--compute running total of GameScore ORDER BY GameScore without partitions
SELECT  * ,
        SUM(GameScore) OVER ( ORDER BY GameScore ) AS RunningTotal
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	RunningTotal
1	4	Name04	Female	1500	3000
2	1	Name01	Male	1500	3000
3	10	Name10	Male	2500	5500
4	2	Name02	Male	2600	8100
5	5	Name05	Female	3350	14800
6	6	Name06	Female	3350	14800
7	9	Name09	Male	3450	18250
8	3	Name03	Male	3500	28750
9	7	Name07	Female	3500	28750
10	8	Name08	Male	3500	28750

```
/*
1.
Output as the following
--Id Name   Gender GameScore RunningTotal ___
--4  Name04 Female 1500      3000           | Duplicate
--1  Name01 Male   1500      3000         __| GameScore
--10 Name10 Male   2500      5500
--2  Name02 Male   2600      8100         ___
--5  Name05 Female 3350      14800          | Duplicate
--6  Name06 Female 3350      14800        __| GameScore
--9  Name09 Male   3450      18250        ___
--3  Name03 Male   3500      28750          | Duplicate
--7  Name07 Female 3500      28750          | GameScore
--8  Name08 Male   3500      28750        __|
2.
--SUM(GameScore) OVER ( ORDER BY GameScore ) AS RunningTotal
OVER ( ORDER BY GameScore ) means ORDER BY GameScore.
Then SELECT SUM(GameScore)   AS RunningTotal
This will compute running total of GameScore without partitions.
If there are some duplicate GameScore,
all the duplicate values will be added to the running total at once.
E.g.
1500+1500=3000, 1500+1500=3000, 1500+1500+2500=5500,
1500+1500+2500+2600=8100,
1500+1500+2500+2600+3350+3350=14800,
1500+1500+2500+2600+3350+3350=14800, ...etc.
*/
```

5.5. Clean up

```
--============================================================
--T034_05_05
--Clean up
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch

============================================================
```

6. Query - NTile Function

```
--============================================================
--T034_06_NtileFunction
--============================================================
/*
1.
Ntile Syntax:
--NTILE (NumberOfGroups) OVER (ORDER BY C1,C2 ...) AS AliasName
--NTILE (NumberOfGroups) OVER (PARTITION BY C1 ORDER BY C1,C2 ...) AS AliasName
1.1.
Divides the rows into a specified NumberOfGroups.
1.2.
If the NumberOfGroups is not divisible,
then the groups will have different sizes.
Larger size groups always come before smaller groups.
E.g.
--NTILE (3) OVER (ORDER BY C1) AS AliasName
NTile function without PARTITION BY
divides 10 rows into 3 Groups .
Group1 size is 4, The size of Group2 and Group3 are 3.
E.g.
--NTILE (2) OVER (ORDER BY C1) AS AliasName
divides 10 rows rows into 2 Groups.
Size of each group is 5
1.3.
NTILE function will try to create as many groups as possible.
If there are 10 rows in the table.
E.g.
--NTILE (11) OVER (ORDER BY C1) AS AliasName
CAN NOT divides 10 rows rows into 11 Groups.
Hense, NTILE (11) divides 10 rows rows into 10 Groups.
1.4.
ORDER BY Clause is compulsory,
PARTITION BY clause is optional
1.5.
--NTILE (NumberOfGroups) OVER (PARTITION BY C1 ORDER BY C2) AS AliasName
NTile function with PARTITION BY :
When the data is partitioned by C1
and then ORDER BY C2,
NTile function creates the NumberOfGroups in each partition.
E.g.
--NTILE(3) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Ntile]
When the data is partitioned by GENDER,
and then ORDER BY GameScore,
NTile function creates 3 groups in each partition.
*/
```

6.1. Create Sample data

```
--============================================================
--T034_06_01
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
```

```
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

6.2. NTILE(3) OVER (ORDER BY GameScore) AS [NTile]

```
--============================================================
--T034_06_02
SELECT  * ,
```

```
        NTILE(3) OVER ( ORDER BY GameScore ) AS [NTile]
FROM    Gamer;
GO -- Run the previous command and begins new batch
/*
--NTILE (3) OVER (ORDER BY C1) AS AliasName
NTile function without PARTITION BY
divides 10 rows into 3 Groups .
Group1 size is 4, The size of Group2 and Group3 are 3.
*/
```

	Id	Name	Gender	GameScore	NTile
1	4	Name04	Female	1500	1
2	1	Name01	Male	1500	1
3	10	Name10	Male	2500	1
4	2	Name02	Male	2600	1
5	5	Name05	Female	3350	2
6	6	Name06	Female	3350	2
7	9	Name09	Male	3450	2
8	3	Name03	Male	3500	3
9	7	Name07	Female	3500	3
10	8	Name08	Male	3500	3

6.3. NTILE(11) OVER (ORDER BY GameScore) AS [NTile]

```
--============================================================
--T034_06_03
SELECT  * ,
        NTILE(11) OVER ( ORDER BY GameScore ) AS [NTile]
FROM    Gamer;
GO -- Run the previous command and begins new batch
/*
NTILE function will try to create as many groups as possible.
If there are 10 rows in the table.
E.g.
--NTILE (11) OVER (ORDER BY C1) AS AliasName
CAN NOT divides 10 rows rows into 11 Groups.
Hense, NTILE (11) divides 10 rows rows into 10 Groups.
*/
```

	Id	Name	Gender	GameScore	NTile
1	4	Name04	Female	1500	1
2	1	Name01	Male	1500	2
3	10	Name10	Male	2500	3
4	2	Name02	Male	2600	4
5	5	Name05	Female	3350	5
6	6	Name06	Female	3350	6
7	9	Name09	Male	3450	7
8	3	Name03	Male	3500	8
9	7	Name07	Female	3500	9
10	8	Name08	Male	3500	10

6.4. NTILE(3) OVER (PARTITION BY Gender ORDER BY GameScore) AS [Ntile]

```
--============================================================
--T034_06_04
SELECT  * ,
        NTILE(3) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Ntile]
FROM    Gamer;
GO -- Run the previous command and begins new batch
/*
--NTILE (NumberOfGroups) OVER (PARTITION BY C1 ORDER BY C2) AS AliasName
NTile function with PARTITION BY :
When the data is partitioned by C1
and then ORDER BY C2,
NTile function creates the NumberOfGroups in each partition.
E.g.
--NTILE(3) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Ntile]
When the data is partitioned by GENDER,
and then ORDER BY GameScore,
NTile function creates 3 groups in each partition.
*/
```

	Id	Name	Gender	GameScore	Ntile
1	4	Name04	Female	1500	1
2	5	Name05	Female	3350	1
3	6	Name06	Female	3350	2
4	7	Name07	Female	3500	3
5	1	Name01	Male	1500	1
6	10	Name10	Male	2500	1
7	2	Name02	Male	2600	2
8	9	Name09	Male	3450	2
9	3	Name03	Male	3500	3
10	8	Name08	Male	3500	3

6.5. Clean up

```
--============================================================
--T034_06_05
--Clean up
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
```

7. Lead(), Lag()

```
--============================================================
--T034_07_LeadFunction_LagFunctions
--============================================================
/*
1.
Lead(), Lag() Syntax :
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName1
--LAG(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName2
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName3
--LAG(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName4
1.1.
ORDER BY C1 is compulsory, PARTITION BY is optional.
----------------------
1.2.
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName1
ORDER BY C1,
LEAD(C1, OffsetNumber, DefaultValue) let you move forward (OffsetNumber) rows.
1.2.1.
That means the value of (currentRow) of LEAD(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow + OffsetNumber) row of C1.
1.2.2.
For the value of last C1 row,
the value of (CurrentRow + OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
1.2.3.
E.g.
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName1
--LEAD(GameScore, 2, -1) OVER ( ORDER BY GameScore ) AS AliasName1
LEAD(GameScore, 2, -1) let you move forward (2) rows.
That means the value of (currentRow) of LEAD(GameScore, 2, -1)
will be the value of (CurrentRow + 2) row of GameScore.
For the value of last GameScore row,
the value of (CurrentRow + 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.
----------------------
1.3.
--LAG(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName2
ORDER BY C1,
LEAD(C1, OffsetNumber, DefaultValue) let you move back forward (OffsetNumber) rows.
1.3.1.
That means the value of (currentRow) of LAG(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow - OffsetNumber) row of C1.
1.3.2.
For the value of First C1 row,
the value of (CurrentRow - OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
1.3.3.
E.g.
--LAG(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName2
--LAG(GameScore, 2, -1) OVER ( ORDER BY GameScore ) AS AliasName2
LAG(GameScore, 2, -1) let you move backforward (2) rows.
That means the value of (currentRow) of LAG(GameScore, 2, -1)
will be the value of (CurrentRow - 2) row of GameScore.
For the value of 1st GameScore row,
the value of (CurrentRow - 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.
----------------------
1.4.
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName3
```

```
PARTITION By C2, and then ORDER BY C1,
LEAD(C1, OffsetNumber, DefaultValue) let you
move forward (OffsetNumber) rows in each PARTITION.
1.4.1.
That means in each PARTITION,
the value of (currentRow) of LEAD(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow + OffsetNumber) row of C1.
1.4.2.
For the value of last C1 row,
the value of (CurrentRow + OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
1.4.3.
E.g.
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName3
--LEAD(GameScore, 2, -1) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS AliasName3
PARTITION By Gender, and then ORDER BY GameScore,
LEAD(GameScore, 2, -1) let you
move forward (2) rows in each PARTITION.
That means in each PARTITION,
the value of (currentRow) of LEAD(GameScore, 2, -1)
will be the value of (CurrentRow + 2) row of GameScore.
For the value of last GameScore row,
the value of (CurrentRow + 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if DefaultValue is specified.
----------------------
1.5.
--LAG(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName4
PARTITION By C2, and then ORDER BY C1,
LAG(C1, OffsetNumber, DefaultValue) let you
move backforward (OffsetNumber) rows in each PARTITION.
1.5.1.
That means in each PARTITION,
the value of (currentRow) of LEAD(C1, OffsetNumber, DefaultValue)
will be the value of (CurrentRow - OffsetNumber) row of C1.
1.5.2.
For the value of first C1 row,
the value of (CurrentRow - OffsetNumber) row of C1
is beyond the table and does not exist.
Thus, it will return NUll or DefaultValue if DefaultValue is specified.
1.5.3.
--LAG(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName4
--LAG(GameScore, 1, -1) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS AliasName4
PARTITION By Gender, and then ORDER BY GameScore,
LAG(GameScore, 1, -1) let you
move backforward (1) rows in each PARTITION.
That means in each PARTITION,
the value of (currentRow) of LEAD(GameScore, 1, -1)
will be the value of (CurrentRow - 1) row of GameScore.
For the value of first GameScore row,
the value of (CurrentRow - 1) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.
*/
```

7.1. Create Sample data

```
--============================================================
--T034_07_01
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
```

```
                WHERE      TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

7.2. LEAD V.S. LAG

```
--============================================================
--T034_07_02
--LEAD V.S. LAG
SELECT  * ,
        LEAD(GameScore, 2, -1) OVER ( ORDER BY GameScore ) AS [LEAD(GameScore,2,-1)] ,
        LAG(GameScore, 1, -1) OVER ( ORDER BY GameScore ) AS [LAG(GameScore,1,-1)]
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	LEAD(GameScore,2,-1)	LAG(GameScore,1,-1)
1	4	Name04	Female	1500	2500	-1
2	1	Name01	Male	1500	2600	1500
3	10	Name10	Male	2500	3350	1500
4	2	Name02	Male	2600	3350	2500
5	5	Name05	Female	3350	3450	2600
6	6	Name06	Female	3350	3500	3350
7	9	Name09	Male	3450	3500	3350
8	3	Name03	Male	3500	3500	3450
9	7	Name07	Female	3500	-1	3500
10	8	Name08	Male	3500	-1	3500

```
/*
1.
Output as the following.
--Id Name   Gender GameScore LEAD(GameScore,2,-1) LAG(GameScore,1,-1)
--4  Name04 Female 1500      2500                 -1
--1  Name01 Male   1500      2600                 1500
--10 Name10 Male   2500      3350                 1500
--2  Name02 Male   2600      3350                 2500
--5  Name05 Female 3350      3450                 2600
--6  Name06 Female 3350      3500                 3350
--9  Name09 Male   3450      3500                 3350
--3  Name03 Male   3500      3500                 3450
--7  Name07 Female 3500      -1                   3500
--8  Name08 Male   3500      -1                   3500
2.
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName1
--LEAD(GameScore, 2, -1) OVER ( ORDER BY GameScore ) AS AliasName1
LEAD(GameScore, 2, -1) let you move forward (2) rows.
That means the value of (currentRow) of LEAD(GameScore, 2, -1)
will be the value of (CurrentRow + 2) row of GameScore.
For the value of last GameScore row,
the value of (CurrentRow + 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.
2.1.
E.g.
When you are on the 1st row, LEAD(Salary, 2, -1)
move forward 2 rows and retrieve the salary from the 3rd row.
E.g.
When you are on the last row, LEAD(Salary, 2, -1)
move forward 2 rows.
Since there no rows beyond the last row,
it returns the default value -1.
3.
--LAG(C1, OffsetNumber, DefaultValue) OVER ( ORDER BY C1 ) AS AliasName2
--LAG(GameScore, 2, -1) OVER ( ORDER BY GameScore ) AS AliasName2
```

```
LAG(GameScore, 2, -1) let you move backforward (2) rows.
That means the value of (currentRow) of LAG(GameScore, 2, -1)
will be the value of (CurrentRow - 2) row of GameScore.
For the value of 1st GameScore row,
the value of (CurrentRow - 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.
3.1.
E.g.
When you are on the last row, LAG(Salary, 1, -1)
move backward 1 row and retrieve the salary from the previous row.
E.g.
When you are on the first row, LAG(Salary, 1, -1)
move backward 1 row.
Since there no rows beyond row 1,
it returns the default value -1.
*/
```

7.3. LEAD V.S. LAG with PARTITION

```
--============================================================
--T034_07_03
--LEAD V.S. LAG with PARTITION
SELECT  * ,
        LEAD(GameScore, 2, -1) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [LEAD(GameScore,2,-1)] ,
        LAG(GameScore, 1, -1) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [LAG(GameScore,1,-1)]
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	LEAD(GameScore,2,-1)	LAG(GameScore,1,-1)
1	4	Name04	Female	1500	3350	-1
2	5	Name05	Female	3350	3500	1500
3	6	Name06	Female	3350	-1	3350
4	7	Name07	Female	3500	-1	3350
5	1	Name01	Male	1500	2600	-1
6	10	Name10	Male	2500	3450	1500
7	2	Name02	Male	2600	3500	2500
8	9	Name09	Male	3450	3500	2600
9	3	Name03	Male	3500	-1	3450
10	8	Name08	Male	3500	-1	3500

```
/*
1.
Output as the following.
--Id Name   Gender GameScore LEAD(GameScore,2,-1) LAG(GameScore,1,-1)
--4  Name04 Female 1500      3350                 -1
--5  Name05 Female 3350      3500                 1500
--6  Name06 Female 3350      -1                   3350
--7  Name07 Female 3500      -1                   3350
--1  Name01 Male   1500      2600                 -1
--10 Name10 Male   2500      3450                 1500
--2  Name02 Male   2600      3500                 2500
--9  Name09 Male   3450      3500                 2600
--3  Name03 Male   3500      -1                   3450
--8  Name08 Male   3500      -1                   3500
2.
--LEAD(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName3
--LEAD(GameScore, 2, -1) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS AliasName3
PARTITION By Gender, and then ORDER BY GameScore,
LEAD(GameScore, 2, -1) let you
```

```
move forward (2) rows in each PARTITION.
That means in each PARTITION,
the value of (currentRow) of LEAD(GameScore, 2, -1)
will be the value of (CurrentRow + 2) row of GameScore.
For the value of last GameScore row,
the value of (CurrentRow + 2) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if DefaultValue is specified.
2.1.
E.g.
When you are on the 1st row of Female, LEAD(Salary, 2, -1)
move forward 2 rows and retrieve the salary from the 3rd row of Female.
E.g.
When you are on the last row of Female, LEAD(Salary, 2, -1)
move forward 2 rows.
Since there no rows beyond the last row,
it returns the default value -1.
3.
--LAG(C1, OffsetNumber, DefaultValue) OVER ( PARTITION BY C2 ORDER BY C1 ) AS AliasName4
--LAG(GameScore, 1, -1) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS AliasName4
PARTITION By Gender, and then ORDER BY GameScore,
LAG(GameScore, 1, -1) let you
move backforward (1) rows in each PARTITION.
That means in each PARTITION,
the value of (currentRow) of LEAD(GameScore, 1, -1)
will be the value of (CurrentRow - 1) row of GameScore.
For the value of first GameScore row,
the value of (CurrentRow - 1) row of GameScore
is beyond the table and does not exist.
Thus, it will return NUll or -1 if -1 is specified.
3.1.
E.g.
When you are on the last row of Female, LAG(Salary, 1, -1)
move backward 1 row and retrieve the salary from the previous row.
E.g.
When you are on the first row of Female, LAG(Salary, 1, -1)
move backward 1 row.
Since there no rows beyond row 1,
it returns the default value -1.
*/
```

7.4. Clean up

```
--============================================================
--T034_07_04
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch


==========================================================
```

8. Query - First_Value()

```
--============================================================
--T034_08_FirstValueFunction
--============================================================
```

```
/*
1.
First_Value() Syntax:
--FIRST_VALUE(C1) OVER ( ORDER BY C2) AS AliasName
--FIRST_VALUE(C1) OVER ( PARTITION BY C3 ORDER BY C2) AS AliasName
1.1.
ORDER BY C1 is compulsory, PARTITION BY is optional.
It returns the first value from the specified column
-------------------------
1.2.
E.g.
--FIRST_VALUE(C1) OVER ( ORDER BY C2) AS AliasName
--FIRST_VALUE([Name]) OVER ( ORDER BY GameScore DESC) AS No1Gamer
FIRST_VALUE() returns the name of the No1Gamer
with highest GameScore from the entire table.
-------------------------
1.3.
E.g.
--FIRST_VALUE(C1) OVER ( PARTITION BY C3 ORDER BY C2) AS AliasName
--FIRST_VALUE([Name]) OVER ( PARTITION BY Gender ORDER BY GameScore DESC) AS No1Gamer
FIRST_VALUE() returns the name of the No1Gamer
with highest GameScore from each PARTITION.
*/
```

8.1. Create Sample data - There are some duplicate GameScore

```
--============================================================
--T034_08_01
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
```

```
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

8.2. FIRST_VALUE(C1) OVER (ORDER BY C2) AS AliasName

```
--============================================================
--T034_08_02
--FIRST_VALUE(C1) OVER ( ORDER BY C2) AS AliasName
SELECT  * ,
        FIRST_VALUE([Name]) OVER ( ORDER BY GameScore DESC ) AS No1Gamer
FROM    Gamer;
GO -- Run the previous command and begins new batch

/*
1.
Output as the following
--Id Name   Gender GameScore No1Gamer
--3  Name03 Male   3500      Name03
--7  Name07 Female 3500      Name03
--8  Name08 Male   3500      Name03
--9  Name09 Male   3450      Name03
--5  Name05 Female 3350      Name03
--6  Name06 Female 3350      Name03
--2  Name02 Male   2600      Name03
--10 Name10 Male   2500      Name03
--1  Name01 Male   1500      Name03
--4  Name04 Female 1500      Name03
2.
E.g.
```

```
--FIRST_VALUE(C1) OVER ( ORDER BY C2) AS AliasName
--FIRST_VALUE([Name]) OVER ( ORDER BY GameScore DESC) AS No1Gamer
FIRST_VALUE() returns the name of the No1Gamer
with highest GameScore from the entire table.
*/
```

	Id	Name	Gender	GameScore	No1Gamer
1	3	Name03	Male	3500	Name03
2	7	Name07	Female	3500	Name03
3	8	Name08	Male	3500	Name03
4	9	Name09	Male	3450	Name03
5	5	Name05	Female	3350	Name03
6	6	Name06	Female	3350	Name03
7	2	Name02	Male	2600	Name03
8	10	Name10	Male	2500	Name03
9	1	Name01	Male	1500	Name03
10	4	Name04	Female	1500	Name03

8.3. FIRST_VALUE(C1) OVER (PARTITION BY C3 ORDER BY C2) AS AliasName

```
--============================================================
--T034_08_03
--FIRST_VALUE(C1) OVER ( PARTITION BY C3 ORDER BY C2) AS AliasName
SELECT  * ,
        FIRST_VALUE([Name]) OVER ( PARTITION BY Gender ORDER BY GameScore DESC ) AS No1Gamer
FROM    Gamer;
GO -- Run the previous command and begins new batch
/*
1.
Output as the following
--Id Name   Gender GameScore No1Gamer
--7  Name07 Female 3500      Name07
--5  Name05 Female 3350      Name07
--6  Name06 Female 3350      Name07
--4  Name04 Female 1500      Name07
--3  Name03 Male   3500      Name03
--8  Name08 Male   3500      Name03
--9  Name09 Male   3450      Name03
--2  Name02 Male   2600      Name03
--10 Name10 Male   2500      Name03
--1  Name01 Male   1500      Name03
2.
E.g.
--FIRST_VALUE(C1) OVER ( PARTITION BY C3 ORDER BY C2) AS AliasName
--FIRST_VALUE([Name]) OVER ( PARTITION BY Gender ORDER BY GameScore DESC) AS No1Gamer
FIRST_VALUE() returns the name of the No1Gamer
with highest GameScore from each PARTITION.
*/
```

	Id	Name	Gender	GameScore	No1Gamer
1	7	Name07	Female	3500	Name07
2	5	Name05	Female	3350	Name07
3	6	Name06	Female	3350	Name07
4	4	Name04	Female	1500	Name07
5	3	Name03	Male	3500	Name03
6	8	Name08	Male	3500	Name03
7	9	Name09	Male	3450	Name03
8	2	Name02	Male	2600	Name03
9	10	Name10	Male	2500	Name03
10	1	Name01	Male	1500	Name03

8.4. Clean up

```
--============================================================
--T034_08_04
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
```

==========

9. Window Functions

```
--============================================================
--T034_09_WindowFunctions
--============================================================
```

9.1. Create Sample data - There are some duplicate GameScore

```
--============================================================
--T034_09_01
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
```

```
        PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

9.2. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

```
--============================================================
--T034_09_02
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
------------------------------------------------
--T034_09_02_01
SELECT  AVG(GameScore)
```

```
FROM    Gamer;
GO -- Run the previous command and begins new batch
/*
AVG(GameScore) will be 2875
*/
```

	(No column name)
1	2875

```
-------------------------------------------
--T034_09_02_02
SELECT  * ,
        AVG(GameScore) OVER ( ORDER BY GameScore ) AS AvgGameScore
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	AvgGameScore
1	4	Name04	Female	1500	1500
2	1	Name01	Male	1500	1500
3	10	Name10	Male	2500	1833
4	2	Name02	Male	2600	2025
5	5	Name05	Female	3350	2466
6	6	Name06	Female	3350	2466
7	9	Name09	Male	3450	2607
8	3	Name03	Male	3500	2875
9	7	Name07	Female	3500	2875
10	8	Name08	Male	3500	2875

```
/*
Output as the following
--Id Name   Gender GameScore AvgGameScore
--4  Name04 Female 1500      1500
--1  Name01 Male   1500      1500
--10 Name10 Male   2500      1833
--2  Name02 Male   2600      2025
--5  Name05 Female 3350      2466
--6  Name06 Female 3350      2466
--9  Name09 Male   3450      2607
--3  Name03 Male   3500      2875
--7  Name07 Female 3500      2875
--8  Name08 Male   3500      2875
The AVG(GameScore) is 2875 which
should be in every row in AvgGameScore column.
However, the actual result is 1500, 1500, 1833, ...etc.
which is not same as we expect.
Because the default for ROWS or RANGE clause is
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
preceding means previous rows.
This means the range includes all previous rows until current row
The 1st row of AVG(GameScore) is 1500
The 2nd row of AVG(GameScore) is (1500+1500)/2=1500
The 3nd row of AVG(GameScore) is (1500+1500+2500)/3=1833
The 4th row of AVG(GameScore) is (1500+1500+2500+2600)/4=2025
....
The 10th row of AVG(GameScore) is (1500+1500+2500+...+3500)/10=2875
*/
```

9.3. AVG(GameScore) OVER (ORDER BY GameScore ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS AvgGameScore

```
--============================================================
--T034_09_03
--AVG(GameScore) OVER ( ORDER BY GameScore ROWS BETWEEN
--    UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS AvgGameScore
SELECT  * ,
        AVG(GameScore) OVER ( ORDER BY GameScore ROWS BETWEEN
        UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS AvgGameScore
FROM    Gamer;
```

	Id	Name	Gender	GameScore	AvgGameScore
1	1	Name01	Male	1500	2875
2	2	Name02	Male	2600	2875
3	3	Name03	Male	3500	2875
4	4	Name04	Female	1500	2875
5	5	Name05	Female	3350	2875
6	6	Name06	Female	3350	2875
7	7	Name07	Female	3500	2875
8	8	Name08	Male	3500	2875
9	9	Name09	Male	3450	2875
10	10	Name10	Male	2500	2875

```
/*
1.
Output as the following
--Id Name   Gender GameScore AvgGameScore
--1  Name01 Male   1500      2875
--2  Name02 Male   2600      2875
--3  Name03 Male   3500      2875
--4  Name04 Female 1500      2875
--5  Name05 Female 3350      2875
--6  Name06 Female 3350      2875
--7  Name07 Female 3500      2875
--8  Name08 Male   3500      2875
--9  Name09 Male   3450      2875
--10 Name10 Male   2500      2875
2.
--AVG(C1) OVER ( ORDER BY C1 ROWS BETWEEN
--UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS AvgC1
UNBOUNDED PRECEDING means all the previous rows.
UNBOUNDED FOLLOWING means all the following rows.
--ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
means from the first row to the last row.
Hense, each rows in AVG(GameScore) will be 2875
*/
```

9.4. ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS AvgGameScore

```
--============================================================
--T034_09_04
--ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING ) AS AvgGameScore
SELECT  * ,
```

```
        AVG(GameScore) OVER ( ORDER BY GameScore
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING ) AS AvgGameScore
FROM    Gamer;
```

	Id	Name	Gender	GameScore	AvgGameScore
1	4	Name04	Female	1500	1500
2	1	Name01	Male	1500	1833
3	10	Name10	Male	2500	2200
4	2	Name02	Male	2600	2816
5	5	Name05	Female	3350	3100
6	6	Name06	Female	3350	3383
7	9	Name09	Male	3450	3433
8	3	Name03	Male	3500	3483
9	7	Name07	Female	3500	3500
10	8	Name08	Male	3500	3500

```
/*
1.
Output as the following
--Id Name   Gender GameScore AvgGameScore
--4  Name04 Female 1500      1500
--1  Name01 Male   1500      1833
--10 Name10 Male   2500      2200
--2  Name02 Male   2600      2816
--5  Name05 Female 3350      3100
--6  Name06 Female 3350      3383
--9  Name09 Male   3450      3433
--3  Name03 Male   3500      3483
--7  Name07 Female 3500      3500
--8  Name08 Male   3500      3500
2.
--ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
1 PRECEDING means 1 previous row.
1 FOLLOWING means 1 following rows.
It means from 1 previous row until 1 following row.
The 1st row of AVG(GameScore) is (1500+1500)/2=1500
The 2nd row of AVG(GameScore) is (1500+1500+2500)/3=1833
The 3nd row of AVG(GameScore) is (1500+2500+2600)/3=2200
The 4th row of AVG(GameScore) is (2500+2600+3350)/3=2816
....
The 10th row of AVG(GameScore) is (3500+3500)/2=3500
*/
```

9.5. Clean up

```
--============================================================
--T034_09_05
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
```

10. Rows And Range

```
--============================================================
--T034_10_RowsAndRange
--============================================================
```

10.1. Create Sample data - There is no duplicate GameScore

```
--============================================================
--T034_10_01
--Create Sample data
--There is no duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 3550 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	3550

10.2. When there is no duplicate GameScore, The following clauses are equivalent

```
--============================================================
--T034_10_02
--When there is no duplicate GameScore
--The following clauses are equivalent:
--T034_10_02_01
SELECT  * ,
        SUM(GameScore) OVER ( ORDER BY GameScore ) AS RunningTotal
FROM    Gamer;
--T034_10_02_02
SELECT  * ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS RunningTotal
FROM    Gamer;
--T034_10_02_03
SELECT  * ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS RunningTotal
FROM    Gamer;
```

	Id	Name	Gender	GameScore	RunningTotal
1	1	Name01	Male	1500	1500
2	2	Name02	Male	2600	4100
3	3	Name03	Male	3500	7600
4	4	Name04	Female	3550	11150

```
/*
1.
Output as the following
--Id Name   Gender GameScore RunningTotal
--1  Name01 Male   1500      1500
--2  Name02 Male   2600      4100
--3  Name03 Male   3500      7600
--4  Name04 Female 3550      11150
2.
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all the previous rows until current row.
*/
```

10.3. Create Sample data - There are some duplicate GameScore

```
--============================================================
--T034_10_03
--Create Sample data
--There are some duplicate GameScore
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
```

```
        PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Male', 2600 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Male	2600

10.4. When there are some duplicate GameScore, Rows and Range treate duplicate differently.

```
--============================================================
--T034_10_04
--When there are some duplicate GameScore
--Rows and Range treat duplicate differently.
SELECT  * ,
        SUM(GameScore) OVER ( ORDER BY GameScore ) AS [Default] ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range] ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
FROM    Gamer;
```

	Id	Name	Gender	GameScore	Default	Range	Rows
1	1	Name01	Male	1500	3000	3000	1500
2	4	Name04	Female	1500	3000	3000	3000
3	5	Name05	Male	2600	8200	8200	5600
4	2	Name02	Male	2600	8200	8200	8200
5	3	Name03	Male	3500	11700	11700	11700

```
/*
1.
Output as the following
--Id Name   Gender GameScore Default Rang  Rows
--1  Name01 Male   1500      3000    3000  1500
--4  Name04 Female 1500      3000    3000  3000
--5  Name05 Male   2600      8200    8200  5600
--2  Name02 Male   2600      8200    8200  8200
--3  Name03 Male   3500      11700   11700 11700
2.
ROWS V.S. RANGE
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all the previous rows until current row.
ROWS and RANGE treat duplicate data differently.
2.1.
--SUM(GameScore) OVER ( ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
ROWS treat duplicates as distinct values.
Thus, SUM(GameScore) will return 1500, 1500+1500=3000,
1500+1500+2600=5600, 1500+1500+2600+2600=8200 ...etc.
2.2.
--SUM(GameScore) OVER ( ORDER BY GameScore
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
RANGE treats them as a single entity.
Thus, SUM(GameScore) will return
1500+1500=3000, 1500+1500=3000
1500+1500+2600+2600=8200, 1500+1500+2600+2600=8200 ...etc.
2.3.
--SUM(GameScore) OVER ( ORDER BY GameScore ) AS [Default]
Default setting is
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
Thus, SUM(GameScore) will return
1500+1500=3000, 1500+1500=3000
1500+1500+2600+2600=8200, 1500+1500+2600+2600=8200 ...etc.
*/
```

10.5. Create Sample data, There are some duplicate GameScore

```
--============================================================
--T034_10_05
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
```

```
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

10.6. There are some duplicate GameScore

```
--============================================================
--T034_10_06
--There are some duplicate GameScore
```

10.6.1. ORDER BY GameScore

```
--------------------------------------------
--T034_10_06_01
--ORDER BY GameScore
SELECT  * ,
              --ORDER BY GameScore
        SUM(GameScore) OVER ( ORDER BY GameScore ) AS [Default] ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range] ,
        SUM(GameScore) OVER ( ORDER BY GameScore
```

```
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
FROM    Gamer;
```

	Id	Name	Gender	GameScore	Default	Range	Rows
1	4	Name04	Female	1500	3000	3000	1500
2	1	Name01	Male	1500	3000	3000	3000
3	10	Name10	Male	2500	5500	5500	5500
4	2	Name02	Male	2600	8100	8100	8100
5	5	Name05	Female	3350	14800	14800	11450
6	6	Name06	Female	3350	14800	14800	14800
7	9	Name09	Male	3450	18250	18250	18250
8	3	Name03	Male	3500	28750	28750	21750
9	7	Name07	Female	3500	28750	28750	25250
10	8	Name08	Male	3500	28750	28750	28750

```
/*
1.
Output as the following
--Id Name   Gender GameScore Default Rang  Rows
--4  Name04 Female 1500      3000    3000  1500
--1  Name01 Male   1500      3000    3000  3000
--10 Name10 Male   2500      5500    5500  5500
--2  Name02 Male   2600      8100    8100  8100
--5  Name05 Female 3350      14800   14800 11450
--6  Name06 Female 3350      14800   14800 14800
--9  Name09 Male   3450      18250   18250 18250
--3  Name03 Male   3500      28750   28750 21750
--7  Name07 Female 3500      28750   28750 25250
--8  Name08 Male   3500      28750   28750 28750
2.
ROWS V.S. RANGE
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all the previous rows until current row.
ROWS and RANGE treat duplicate data differently.
2.1.
--SUM(GameScore) OVER ( ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
ROWS treat duplicates as distinct values.
Thus, SUM(GameScore) will return 1500, 1500+1500=3000,
1500+1500+2500=5500, 1500+1500+2500+2600=8100 ...etc.
2.2.
--SUM(GameScore) OVER ( ORDER BY GameScore
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
RANGE treats them as a single entity.
Thus, SUM(GameScore) will return
1500+1500=3000, 1500+1500=3000,
1500+1500+2500=5500, 1500+1500+2500+2600=8100,
1500+1500+2500+2600+3350+3350=14800,
1500+1500+2500+2600+3350+3350=14800,...etc
2.3.
--SUM(GameScore) OVER ( ORDER BY GameScore ) AS [Default]
Default setting is
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
RANGE treats them as a single entity.
Thus, SUM(GameScore) will return
1500+1500=3000, 1500+1500=3000,
1500+1500+2500=5500, 1500+1500+2500+2600=8100,
1500+1500+2500+2600+3350+3350=14800,
1500+1500+2500+2600+3350+3350=14800,...etc
*/
```

10.6.2. PARTITION BY Gender ORDER BY GameScore

```
-----------------------------------------------
--T034_10_06_02
--PARTITION BY Gender ORDER BY GameScore
SELECT  * ,
                --PARTITION BY Gender ORDER BY GameScore
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Default] ,
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range] ,
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
FROM    Gamer;
```

	Id	Name	Gender	GameScore	Default	Range	Rows
1	4	Name04	Female	1500	1500	1500	1500
2	5	Name05	Female	3350	8200	8200	4850
3	6	Name06	Female	3350	8200	8200	8200
4	7	Name07	Female	3500	11700	11700	11700
5	1	Name01	Male	1500	1500	1500	1500
6	10	Name10	Male	2500	4000	4000	4000
7	2	Name02	Male	2600	6600	6600	6600
8	9	Name09	Male	3450	10050	10050	10050
9	3	Name03	Male	3500	17050	17050	13550
10	8	Name08	Male	3500	17050	17050	17050

```
/*
1.
Output as the following
--Id Name   Gender GameScore Default Rang  Rows
--4  Name04 Female 1500      1500    1500  1500
--5  Name05 Female 3350      8200    8200  4850
--6  Name06 Female 3350      8200    8200  8200
--7  Name07 Female 3500      11700   11700 11700
--1  Name01 Male   1500      1500    1500  1500
--10 Name10 Male   2500      4000    4000  4000
--2  Name02 Male   2600      6600    6600  6600
--9  Name09 Male   3450      10050   10050 10050
--3  Name03 Male   3500      17050   17050 13550
--8  Name08 Male   3500      17050   17050 17050
2.
ROWS V.S. RANGE
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all the previous rows until current row.
ROWS and RANGE treat duplicate data differently.
2.1.
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows]
ROWS treat duplicates as distinct values.
Thus,
SUM(GameScore) for Female will return
1500, 1500+3350=4850, 1500+3350+3350=8200,
1500+3350+3350+3500=11700
SUM(GameScore) for Male will return
1500, 1500+2500=4000, 1500+2500+2600=6600,
1500+2500+2600+3450=10050,
1500+2500+2600+3450+3500=13550,
1500+2500+2600+3450+3500+3500=17050
2.2.
```

```
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
RANGE treats them as a single entity.
Thus,
SUM(GameScore) for Female will return
1500, 1500+3350+3350=8200, 1500+3350+3350=8200,
1500+3350+3350+3500=11700
SUM(GameScore) for Male will return
1500, 1500+2500=4000, 1500+2500+2600=6600,
1500+2500+2600+3450=10050,
1500+2500+2600+3450+3500+3500=17050,
1500+2500+2600+3450+3500+3500=17050
2.3.
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Default]
Default setting is
--SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range]
RANGE treats them as a single entity.
Thus,
SUM(GameScore) for Female will return
1500, 1500+3350+3350=8200, 1500+3350+3350=8200,
1500+3350+3350+3500=11700
SUM(GameScore) for Male will return
1500, 1500+2500=4000, 1500+2500+2600=6600,
1500+2500+2600+3450=10050,
1500+2500+2600+3450+3500+3500=17050,
1500+2500+2600+3450+3500+3500=17050
*/
```

10.6.3. Logic Error - (ORDER BY GameScore) With (PARTITION BY Gender ORDER BY GameScore)

```
-----------------------------------------------
--T034_10_06_03
--Logic Error :
--ORDER BY GameScore
--with
--PARTITION BY Gender ORDER BY GameScore
SELECT  * ,
              --ORDER BY GameScore
        SUM(GameScore) OVER ( ORDER BY GameScore ) AS [Default] ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range] ,
        SUM(GameScore) OVER ( ORDER BY GameScore
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows] ,
              --PARTITION BY Gender ORDER BY GameScore
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore ) AS [Default2] ,
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Range2] ,
        SUM(GameScore) OVER ( PARTITION BY Gender ORDER BY GameScore
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Rows2]
FROM    Gamer;
```

	Id	Name	Gender	GameScore	Default	Range	Rows	Default2	Range2	Rows2
1	4	Name04	Female	1500	3000	3000	1500	1500	1500	1500
2	5	Name05	Female	3350	14800	14800	11450	8200	8200	4850
3	6	Name06	Female	3350	14800	14800	14800	8200	8200	8200
4	7	Name07	Female	3500	28750	28750	25250	11700	11700	11700
5	1	Name01	Male	1500	3000	3000	3000	1500	1500	1500
6	10	Name10	Male	2500	5500	5500	5500	4000	4000	4000
7	2	Name02	Male	2600	8100	8100	8100	6600	6600	6600
8	9	Name09	Male	3450	18250	18250	18250	10050	10050	10050
9	3	Name03	Male	3500	28750	28750	21750	17050	17050	13550
10	8	Name08	Male	3500	28750	28750	28750	17050	17050	17050

```
/*
1.
Output as the following
--Id Name   Gender GameScore Default Rang  Rows  Default2 Rang2  Rows2
--4  Name04 Female 1500      3000    3000  1500  1500     1500   1500
--5  Name05 Female 3350      14800   14800 11450 8200     8200   4850
--6  Name06 Female 3350      14800   14800 14800 8200     8200   8200
--7  Name07 Female 3500      28750   28750 25250 11700    11700  11700
--1  Name01 Male   1500      3000    3000  3000  1500     1500   1500
--10 Name10 Male   2500      5500    5500  5500  4000     4000   4000
--2  Name02 Male   2600      8100    8100  8100  6600     6600   6600
--9  Name09 Male   3450      18250   18250 18250 10050    10050  10050
--3  Name03 Male   3500      28750   28750 21750 17050    17050  13550
--8  Name08 Male   3500      28750   28750 28750 17050    17050  17050
2.
When you using
--ORDER BY GameScore
--with
--PARTITION BY Gender ORDER BY GameScore
The "ORDER BY GameScore" result will become very strange.
The "PARTITION BY Gender ORDER BY GameScore" result will be same as expect.
*/
```

==========

11. Last Value Function

```
--============================================================
--T034_11_LastValueFunction
--============================================================
/*
Last_Value() Syntax:
--LAST_VALUE(C1) OVER ( ORDER BY C2) AS AliasName
--LAST_VALUE(C1) OVER ( PARTITION BY C3 ORDER BY C2) AS AliasName
8.1.
ORDER BY C1 is compulsory, PARTITION BY is optional.
It returns the last value from the specified column
--------------------------
8.2.
E.g.
--LAST_VALUE(C1) OVER ( ORDER BY C2) AS AliasName
--LAST_VALUE([Name]) OVER ( ORDER BY GameScore) AS No1Gamer
LAST_VALUE() returns the name of the No1Gamer
with highest GameScore from the entire table.
--------------------------
8.3.
E.g.
--LAST_VALUE(C1) OVER ( PARTITION BY C3 ORDER BY C2) AS AliasName
--LAST_VALUE([Name]) OVER ( PARTITION BY Gender ORDER BY GameScore) AS No1Gamer
```

```
LAST_VALUE() returns the name of the No1Gamer
with highest GameScore from each PARTITION.
*/
```

11.1. Create Sample data - There are some duplicate GameScore

```
--============================================================
--T034_11_01
--Create Sample data
--There are some duplicate GameScore
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
        PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'Name01', 'Male', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name02', 'Male', 2600 );
INSERT  INTO Gamer
VALUES  ( 'Name03', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name04', 'Female', 1500 );
INSERT  INTO Gamer
VALUES  ( 'Name05', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name06', 'Female', 3350 );
INSERT  INTO Gamer
VALUES  ( 'Name07', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name08', 'Male', 3500 );
INSERT  INTO Gamer
VALUES  ( 'Name09', 'Male', 3450 );
INSERT  INTO Gamer
VALUES  ( 'Name10', 'Male', 2500 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore
1	1	Name01	Male	1500
2	2	Name02	Male	2600
3	3	Name03	Male	3500
4	4	Name04	Female	1500
5	5	Name05	Female	3350
6	6	Name06	Female	3350
7	7	Name07	Female	3500
8	8	Name08	Male	3500
9	9	Name09	Male	3450
10	10	Name10	Male	2500

11.2. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW : The following clauses are equivalent

```
--============================================================
--T034_11_02
--The following clauses are equivalent:
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
--T034_11_02_01
SELECT  * ,
        LAST_VALUE([Name]) OVER ( ORDER BY GameScore
              RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Last]
FROM    Gamer;
GO -- Run the previous command and begins new batch
------------------------------
--T034_11_02_02
SELECT  * ,
        LAST_VALUE([Name]) OVER ( ORDER BY GameScore ) AS [Last]
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	Last
1	4	Name04	Female	1500	Name01
2	1	Name01	Male	1500	Name01
3	10	Name10	Male	2500	Name10
4	2	Name02	Male	2600	Name02
5	5	Name05	Female	3350	Name06
6	6	Name06	Female	3350	Name06
7	9	Name09	Male	3450	Name09
8	3	Name03	Male	3500	Name08
9	7	Name07	Female	3500	Name08
10	8	Name08	Male	3500	Name08

```
/*
1.
Output as the following
```

```
--Id Name   Gender GameScore Last
--4  Name04 Female 1500      Name01
--1  Name01 Male   1500      Name01
--10 Name10 Male   2500      Name10
--2  Name02 Male   2600      Name02
--5  Name05 Female 3350      Name06
--6  Name06 Female 3350      Name06
--9  Name09 Male   3450      Name09
--3  Name03 Male   3500      Name08
--7  Name07 Female 3500      Name08
--8  Name08 Male   3500      Name08
2.
--LAST_VALUE([Name]) OVER ( ORDER BY GameScore ) AS [Last]
2.1.
ORDER BY GameScore, each row of [Last]
will take the value of last [Name]
from all previous rows to current row.
2.2.
The default setting is
--RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all previous to current row.
ROWS and RANGE treat duplicate data differently.
RANGE treats them as a single entity.
Thus,
The 1st, 2nd row of LAST_VALUE(Name),
because Name04 and Name01 both have GameScore 1500, Hense both return Name01.
The 3nd row of LAST_VALUE(Name) returns Name10
The 4th row of LAST_VALUE(Name) returns Name02
The 5th, 6th row of LAST_VALUE(Name),
because Name05 and Name06 both have GameScore 3350, Hense both return Name06.
The 7th row of LAST_VALUE(Name) returns Name09
The 8th, 9th, 10th row of LAST_VALUE(Name),
because Name03, Name07, Name08 have GameScore 3500, Hense both return Name08.
*/
```

11.3. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

```
--============================================================
--T034_11_03
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
SELECT  * ,
        LAST_VALUE([Name]) OVER ( ORDER BY GameScore
             ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Last]
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	Last
1	4	Name04	Female	1500	Name04
2	1	Name01	Male	1500	Name01
3	10	Name10	Male	2500	Name10
4	2	Name02	Male	2600	Name02
5	5	Name05	Female	3350	Name05
6	6	Name06	Female	3350	Name06
7	9	Name09	Male	3450	Name09
8	3	Name03	Male	3500	Name03
9	7	Name07	Female	3500	Name07
10	8	Name08	Male	3500	Name08

```
/*
1.
```

```
Output as the following
--Id Name   Gender GameScore Last
--4  Name04 Female 1500      Name04
--1  Name01 Male   1500      Name01
--10 Name10 Male   2500      Name10
--2  Name02 Male   2600      Name02
--5  Name05 Female 3350      Name05
--6  Name06 Female 3350      Name06
--9  Name09 Male   3450      Name09
--3  Name03 Male   3500      Name03
--7  Name07 Female 3500      Name07
--8  Name08 Male   3500      Name08
2.
--LAST_VALUE([Name]) OVER ( ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS [Last]
2.1.
ORDER BY GameScore, each row of [Last]
will take the value of last row of [Name]
from all previous rows to current row.
2.2.
--BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
means from all previous to current row.
ROWS and RANGE treat duplicate data differently.
ROWS treat duplicates as distinct values.
Thus,
The 1st row of LAST_VALUE(Name) returns Name04.
The 2nd row of LAST_VALUE(Name) returns Name01,
even when Name04 and Name01 both have the same GameScore 1500.
...etc.
*/
```

11.4. ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

```
--============================================================
--T034_11_04
--ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
SELECT  * ,
        LAST_VALUE([Name]) OVER ( ORDER BY GameScore
             ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS [Last]
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	Last
1	4	Name04	Female	1500	Name08
2	1	Name01	Male	1500	Name08
3	10	Name10	Male	2500	Name08
4	2	Name02	Male	2600	Name08
5	5	Name05	Female	3350	Name08
6	6	Name06	Female	3350	Name08
7	9	Name09	Male	3450	Name08
8	3	Name03	Male	3500	Name08
9	7	Name07	Female	3500	Name08
10	8	Name08	Male	3500	Name08

```
/*
1.
Output as the following
--Id Name   Gender GameScore Last
--4  Name04 Female 1500      Name08
```

```
--1  Name01 Male   1500      Name08
--10 Name10 Male   2500      Name08
--2  Name02 Male   2600      Name08
--5  Name05 Female 3350      Name08
--6  Name06 Female 3350      Name08
--9  Name09 Male   3450      Name08
--3  Name03 Male   3500      Name08
--7  Name07 Female 3500      Name08
--8  Name08 Male   3500      Name08
2.
--LAST_VALUE([Name]) OVER ( ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS [Last]
2.1.
ORDER BY GameScore, each row of [Last]
will take the value of last row of [Name]
from all previous rows unit all followings row.
2.2.
--BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
meansfrom all previous rows unit all followings row.
ROWS and RANGE treat duplicate data differently.
ROWS treat duplicates as distinct values.
Name08 is the value of last [Last]
Thus,
The 1st row of LAST_VALUE(Name) returns Name08.
The 2nd row of LAST_VALUE(Name) returns Name08,
...etc.
*/
```

11.5. PARTITION BY C2 ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

```
--============================================================
--T034_11_05
--PARTITION BY C2 ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
SELECT  * ,
        LAST_VALUE([Name]) OVER ( PARTITION BY Gender ORDER BY GameScore
             ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS [Last]
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	Last
1	4	Name04	Female	1500	Name07
2	5	Name05	Female	3350	Name07
3	6	Name06	Female	3350	Name07
4	7	Name07	Female	3500	Name07
5	1	Name01	Male	1500	Name08
6	10	Name10	Male	2500	Name08
7	2	Name02	Male	2600	Name08
8	9	Name09	Male	3450	Name08
9	3	Name03	Male	3500	Name08
10	8	Name08	Male	3500	Name08

```
/*
1.
Output as the following
--Id Name   Gender GameScore Last
--4  Name04 Female 1500      Name07
--5  Name05 Female 3350      Name07
--6  Name06 Female 3350      Name07
```

```
--7  Name07 Female 3500      Name07
--1  Name01 Male   1500      Name08
--10 Name10 Male   2500      Name08
--2  Name02 Male   2600      Name08
--9  Name09 Male   3450      Name08
--3  Name03 Male   3500      Name08
--8  Name08 Male   3500      Name08
2.
--LAST_VALUE([Name]) OVER ( ORDER BY GameScore
--ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS [Last]
2.1.
ORDER BY GameScore, each row of [Last]
will take the value of last row of [Name]
from all previous rows unit all followings row.
2.2.
--BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
meansfrom all previous rows unit all followings row.
ROWS and RANGE treat duplicate data differently.
ROWS treat duplicates as distinct values.
Name07 is the value of last [Last] for Female
Name08 is the value of last [Last] for Male
Thus,
The 1st row of LAST_VALUE(Name) returns Name07.
...
The 4th row of LAST_VALUE(Name) returns Name07,
The 5th row of LAST_VALUE(Name) returns Name08,
...
The 10th row of LAST_VALUE(Name) returns Name08,
*/
```

11.6. PARTITION BY C2 ORDER BY C1 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

```
--============================================================
--T034_11_06
--Clean up
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch

================================================================
```

12. Find Nth highest GameScore

```
--============================================================
--T034_12_FindNthHighestSalary
--============================================================
```

12.1. Create Sample Data

```
--============================================================
--T034_12_01
--Create Sample Data
IF ( EXISTS ( SELECT    *
```

```
                FROM      INFORMATION_SCHEMA.TABLES
                WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  FirstName NVARCHAR(50) ,
  LastName NVARCHAR(50) ,
  Gender NVARCHAR(50) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'AFirst01', 'XLast01', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'AFirst02', 'YLast02', 'Female', 4000 );
INSERT  INTO Gamer
VALUES  ( 'BFirst03', 'YLast03', 'Male', 4600 );
INSERT  INTO Gamer
VALUES  ( 'BFirst04', 'YLast04', 'Male', 5400 );
INSERT  INTO Gamer
VALUES  ( 'BFirst05', 'ZLast05', 'Female', 5400 );
INSERT  INTO Gamer
VALUES  ( 'CFirst06', 'YLast06', 'Male', 4000 );
INSERT  INTO Gamer
VALUES  ( 'CFirst07', 'YLast07', 'Male', 4400 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
GO -- Run the previous command and begins new batch
```

	Id	FirstName	LastName	Gender	GameScore
1	1	AFirst01	XLast01	Female	3500
2	2	AFirst02	YLast02	Female	4000
3	3	BFirst03	YLast03	Male	4600
4	4	BFirst04	YLast04	Male	5400
5	5	BFirst05	ZLast05	Female	5400
6	6	CFirst06	YLast06	Male	4000
7	7	CFirst07	YLast07	Male	4400

12.2. Get the highest GameScore

```
--============================================================
--T034_12_02
--Get the highest GameScore
SELECT  MAX(GameScore)
FROM    Gamer;
GO -- Run the previous command and begins new batch
--5400
```

12.3. Get the 2nd highest GameScore

```
--============================================================
--T034_12_03
--Get the 2nd highest GameScore
SELECT  MAX(GameScore)
FROM    Gamer
WHERE   GameScore < ( SELECT    MAX(GameScore)
                      FROM      Gamer
                    );
GO -- Run the previous command and begins new batch
--4600
--SubQuery always run first.
```

12.4. Get the Nth highest GameScore by subQuery

```
--============================================================
--T034_12_04
--Get the Nth highest GameScore by subQuery
SELECT TOP 1
        GameScore
FROM    ( --SELECT DISTINCT TOP N  --N means Nth highest GameScore
          SELECT DISTINCT TOP 2
                  GameScore
          FROM      Gamer
          ORDER BY  GameScore DESC
        ) RESULT
ORDER BY GameScore;
--4600
--SubQuery always run first.
```

12.5. Revise RANK() OVER (ORDER BY C1, C2, ...) / DENSE_RANK() OVER (ORDER BY C1, C2, ...) / ROW_NUMBER() OVER (ORDER BY C1, C2, ...)

```
--============================================================
--T034_12_05
--Revise
--RANK() OVER (ORDER BY C1, C2, ...)
--DENSE_RANK() OVER (ORDER BY C1, C2, ...)
--ROW_NUMBER() OVER (ORDER BY C1, C2, ...)
SELECT  FirstName ,
        LastName ,
        GameScore ,
        Gender ,
        RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank] ,
        DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank ,
        ROW_NUMBER() OVER ( ORDER BY GameScore DESC ) AS RowNumber
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	FirstName	LastName	GameScore	Gender	Rank	DenseRank	RowNumber
1	BFirst04	YLast04	5400	Male	1	1	1
2	BFirst05	ZLast05	5400	Female	1	1	2
3	BFirst03	YLast03	4600	Male	3	2	3
4	CFirst07	YLast07	4400	Male	4	3	4
5	CFirst06	YLast06	4000	Male	5	4	5
6	AFirst02	YLast02	4000	Female	5	4	6
7	AFirst01	XLast01	3500	Female	7	5	7

```
/*
1.
Output as following.
--BFirst04   YLast04      5400   Male   1      1      1
--BFirst05   ZLast05      5400   Female 1      1      2
--BFirst03   YLast03      4600   Male   3      2      3
--CFirst07   YLast07      4400   Male   4      3      4
--CFirst06   YLast06      4000   Male   5      4      5
--AFirst02   YLast02      4000   Female 5      4      6
--AFirst01   XLast01      3500   Female 7      5      7
2.
--RANK() OVER ( ORDER BY GameScore DESC ) AS [Rank]
--DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DenseRank
Both RANK() and DENSE_RANK() returns
the sequential Rank number by GameScore and it starts from 1.
Rank function skips ranking(s) if there is a tie (平局)
E.g. RANK() returns 1, 1, 3, 4, 5
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. DENSE_RANK() returns 1, 1, 2, 3, 4
3.
Compare ROW_NUMBER(), RANK(), and DENSE_RANK()
3.1.
If there is no duplicate data row,
there is no different in ROW_NUMBER(), RANK(), or DENSE_RANK()
3.2.
If there are some duplicate data rows,
All ROW_NUMBER(), RANK(), and DENSE_RANK() return
an increasing unique number for each row starting at 1.
3.2.1.
ROW_NUMBER() still returns different Row Number
if it meets duplicate data rows.
E.g. 1,2,3,4,5,6,7,8,9,10
3.2.2.
Both RANK() and DENSE_RANK() return the same Rank Number
if it meets duplicate data rows.
However,
Rank function skips ranking(s) if there is a tie (平局).
E.g. 1,1,1,4,5,5,7,8,9,9
Rank function will NOT skip ranking(s) if there is a tie (平局)
E.g. 1,1,1,2,3,3,4,5,6,6
*/
```

12.6. Get the Nth highest GameScore by CTE and DENSE_RANK()

```
--============================================================
--T034_12_06
--Get the Nth highest GameScore by CTE and DENSE_RANK()
WITH    GamerDenseRankCTE
          AS ( SELECT    GameScore ,
                         DENSE_RANK() OVER ( ORDER BY GameScore DESC ) AS DENSERANK
               FROM      Gamer
             )
```

```
    SELECT TOP 1
            GameScore
    FROM    GamerDenseRankCTE
    WHERE   DENSERANK = 2;
--  WHERE   DENSERANK = N;  --N means Nth highest GameScore
GO -- Run the previous command and begins new batch
--4600
```

12.7. Get the Nth highest GameScore by CTE and ROW_NUMBER()

```
--=============================================================
--T034_12_07
--Get the Nth highest GameScore by CTE and ROW_NUMBER()
WITH    GamerRowNumberCTE
          AS ( SELECT   GameScore ,
                        ROW_NUMBER() OVER ( ORDER BY GameScore DESC ) AS RowNumber
               FROM     Gamer
             )
    SELECT TOP 1
            GameScore
    FROM    GamerRowNumberCTE
    WHERE   RowNumber = 2;
--  WHERE   RowNumber = N;  --N means Nth highest GameScore
GO -- Run the previous command and begins new batch
/*
Return 5400, but the 2nd highest GameScore is actually 4600.
It is because there are two Gamers get No1 GameScore 5400.
This ROW_NUMBER way can only work when there is no duplicates.
*/
```

	GameScore
1	5400

12.8. Clean up

```
--=============================================================
--T034_12_08
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
```