COVID-19

Prepared For: CS527

Prepared By: Josh Levine, Pranav Shivkumar, Pratik Mistry, Shounak Rangwala, Swapnil Kamate, Vikhyat Dhamija

Task 1- Uploading of COVID-19.csv to DBMS

COVID-19.csv file was successfully uploaded to the DBMS - SQL SERVER.

Sample Data From Covid Table

Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities
1	NULL	Afghanistan	2020-01-22 ...	0	0
2	NULL	Afghanistan	2020-01-23 ...	0	0
3	NULL	Afghanistan	2020-01-24 ...	0	0
4	NULL	Afghanistan	2020-01-25 ...	0	0
5	NULL	Afghanistan	2020-01-26 ...	0	0
6	NULL	Afghanistan	2020-01-27 ...	0	0
7	NULL	Afghanistan	2020-01-28 ...	0	0
8	NULL	Afghanistan	2020-01-29 ...	0	0
9	NULL	Afghanistan	2020-01-30 ...	0	0
10	NULL	Afghanistan	2020-01-31 ...	0	0
11	NULL	Afghanistan	2020-02-01 ...	0	0
12	NULL	Afghanistan	2020-02-02 ...	0	0
13	NULL	Afghanistan	2020-02-03 ...	0	0
14	NULL	Afghanistan	2020-02-04 ...	0	0
15	NULL	Afghanistan	2020-02-05 ...	0	0
16	NULL	Afghanistan	2020-02-06 ...	0	0
17	NULL	Afghanistan	2020-02-07 ...	0	0
18	NULL	Afghanistan	2020-02-08 ...	0	0
19	NULL	Afghanistan	2020-02-09 ...	0	0
20	NULL	Afghanistan	2020-02-10 ...	0	0
21	NULL	Afghanistan	2020-02-11 ...	0	0
22	NULL	Afghanistan	2020-02-12 ...	0	0

TASK 2:- Using Windowing/Partition to transform the “Cumulative numbers” to “Daily Numbers”

- The SQL query :

  *SELECT * , ConfirmedCases- LAG(ConfirmedCases,1) OVER(Partition By Country_Region,Province_State ORDER BY Date) AS "Confirmed Daily" ,Fatalities-LAG(Fatalities,1) OVER(Partition By Country_Region,Province_State ORDER BY Date) As "Fatalities Daily" FROM dbo.COVID*

  Important SQL Clauses and Functions Used

- The PARTITION BY clause is used to divide a query’s result set into partitions, after which the WINDOW function operates on each partition separately and recalculates for each partition.
- LAG() is a window function that provides access to a row at a specified physical offset which comes before the current row.
- ***NOTE**: Window Function is not supported on RDS MySQL below version 8*

Sample Result - Task 2

Id	Province_State	Country_Region	Date	ConfirmedCases	Fatalities	Confirmed Daily	Fatalities Daily
29480	South Dakota	US	2020-03-16 ...	10	1	1	0
29481	South Dakota	US	2020-03-17 ...	11	1	1	0
29482	South Dakota	US	2020-03-18 ...	11	1	0	0
29483	South Dakota	US	2020-03-19 ...	11	1	0	0
29484	South Dakota	US	2020-03-20 ...	14	1	3	0
29485	South Dakota	US	2020-03-21 ...	14	1	0	0
29486	South Dakota	US	2020-03-22 ...	21	1	7	0
29487	South Dakota	US	2020-03-23 ...	28	1	7	0
29488	South Dakota	US	2020-03-24 ...	30	1	2	0
29489	South Dakota	US	2020-03-25 ...	41	1	11	0
29490	South Dakota	US	2020-03-26 ...	46	1	5	0
29491	South Dakota	US	2020-03-27 ...	58	1	12	0
29492	South Dakota	US	2020-03-28 ...	68	1	10	0
29493	South Dakota	US	2020-03-29 ...	90	1	22	0
29494	South Dakota	US	2020-03-30 ...	101	1	11	0
29495	South Dakota	US	2020-03-31 ...	108	1	7	0
29496	South Dakota	US	2020-04-01 ...	129	2	21	1
29497	South Dakota	US	2020-04-02 ...	165	2	36	0
29498	South Dakota	US	2020-04-03 ...	187	2	22	0
29499	South Dakota	US	2020-04-04 ...	212	2	25	0
29533	Tennessee	US	2020-01-22 ...	0	0	NULL	NULL
29534	Tennessee	US	2020-01-23 ...	0	0	0	0
29535	Tennessee	US	2020-01-24 ...	0	0	0	0
29536	Tennessee	US	2020-01-25 ...	0	0	0	0
29537	Tennessee	US	2020-01-26 ...	0	0	0	0
29538	Tennessee	US	2020-01-27 ...	0	0	0	0
29539	Tennessee	US	2020-01-28 ...	0	0	0	0

Task-3 Aggregate "ConfirmedDaily" and "FatalitiesDaily" by "Country_Region and WeekOfYear(Date)"

The SQL query is as follows:

SELECT T.Country_Region, T.WeekOfYear, sum(T."Fatalities Daily") as "FatalitiesDaily", sum(T."Confirmed Daily") as "ConfirmedDaily" into COVID_19_aggr from (SELECT Country_Region ,DATEPART(ww,Date) As WeekOfYear, ConfirmedCases- LAG(ConfirmedCases,1) OVER(Partition By Country_Region, Province_State ORDER BY Date) AS "Confirmed Daily" , Fatalities-LAG(Fatalities,1) OVER(Partition By Country_Region,Province_State ORDER BY Date) As "Fatalities Daily" FROM dbo.COVID) as T group by T.Country_Region, T.WeekOfYear order by T.Country_Region, T.WeekOfYear

Important SQL Clauses and Functions Used

DATEPART: The DATEPART() function returns an integer which is a part of a date such as a day, month, and year.

Sample Result - Task 3

	Country_Region	WeekOfYear	FatalitiesDaily	ConfirmedDaily
1	Afghanistan	4	0	0
2	Afghanistan	5	0	0
3	Afghanistan	6	0	0
4	Afghanistan	7	0	0
5	Afghanistan	8	0	0
6	Afghanistan	9	0	1
7	Afghanistan	10	0	0
8	Afghanistan	11	0	10
9	Afghanistan	12	0	13
10	Afghanistan	13	4	86
11	Afghanistan	14	3	189
12	Albania	4	0	0
13	Albania	5	0	0
14	Albania	6	0	0
15	Albania	7	0	0
16	Albania	8	0	0
17	Albania	9	0	0
18	Albania	10	0	0
19	Albania	11	1	38
20	Albania	12	1	38
21	Albania	13	8	121
22	Albania	14	10	136
23	Algeria	4	0	0
24	Algeria	5	0	0
25	Algeria	6	0	0
26	Algeria	7	0	0
27	Algeria	8	0	0
28	Algeria	9	0	1
29	Algeria	10	0	16

Task 4 : Use "GROUP BY CUBE", "GROUP BY ROLLUP", and "GROUPING SETS" against "COVID_19_aggr" table.

GROUP BY CUBE

This statement creates groups for all possible combinations of the specified columns.The syntax for the GROUP BY CUBE query is:

SELECT d1, d2, aggregate_function(c4) FROM table_name GROUP BY CUBE (d1, d2);

Query:

select Country_Region, WeekOfYear, SUM(ConfirmedDaily) as ConfirmedDaily , SUM(FatalitiesDaily) as FatalitiesDaily from COVID_19_aggr group by cube (Country_Region,WeekOfYear)

GROUP BY ROLLUP

This statement creates groups of all possible combinations of the specified columns and ‘rolls up’ or provides the grand total and subtotal from the results. The syntax for the GROUP BY ROLLUP query is:

SELECT d1, d2, d3, aggregate_function(c4) FROM table_name GROUP BY ROLLUP (d1, d2, d3);

Query:

Select Country_Region, WeekOfYear, SUM(ConfirmedDaily) as ConfirmedDaily , SUM(FatalitiesDaily) as FatalitiesDaily from COVID_19_aggr group by rollup(Country_Region,WeekOfYear)

Sample Result - Task 4(Group By CUBE)

	Country_Region	WeekOfYear	ConfirmedDaily	FatalitiesDaily
1986	Venezuela	14	36	5
1987	Vietnam	14	66	0
1988	West Bank and Gaza	14	119	0
1989	Zambia	14	11	1
1990	Zimbabwe	14	2	0
1991	NULL	14	536684	33953
1992	NULL	NULL	1196671	64584
1993	Afghanistan	NULL	299	7
1994	Albania	NULL	333	20
1995	Algeria	NULL	1251	130
1996	Andorra	NULL	466	17
1997	Angola	NULL	10	2
1998	Antigua and Barbuda	NULL	15	0
1999	Argentina	NULL	1451	43
2000	Armenia	NULL	770	7
2001	Australia	NULL	5550	30
2002	Austria	NULL	11781	186

Sample Result - Task 4(Group By RollUP)

	Country_Region	WeekOfYear	ConfirmedDaily	FatalitiesDaily
2142	Zambia	9	0	0
2143	Zambia	10	0	0
2144	Zambia	11	0	0
2145	Zambia	12	2	0
2146	Zambia	13	26	0
2147	Zambia	14	11	1
2148	Zambia	NULL	39	1
2149	Zimbabwe	4	0	0
2150	Zimbabwe	5	0	0
2151	Zimbabwe	6	0	0
2152	Zimbabwe	7	0	0
2153	Zimbabwe	8	0	0
2154	Zimbabwe	9	0	0
2155	Zimbabwe	10	0	0
2156	Zimbabwe	11	0	0
2157	Zimbabwe	12	3	0
2158	Zimbabwe	13	4	1
2159	Zimbabwe	14	2	0
2160	Zimbabwe	NULL	9	1
2161	NULL	NULL	1196671	64584

Task 4 (Continued): using GROUPING SETS

This statement provides the option to combine multiple GROUP BY clauses into a single clause. For columns c1, c2, c3 of a table, the query GROUP BY ROLLUP(c1, c2, c3) returns the same result as GROUP BY GROUPING SETS (ROLLUP(c1, c2, c3))

The SQL Query is :

select Country_Region, WeekOfYear, SUM(ConfirmedDaily) as ConfirmedDaily , SUM(FatalitiesDaily) as FatalitiesDaily from COVID_19_aggr group by GROUPING SETS ((Country_Region), (WeekOfYear), ())

Sample Result - Task 4(Grouping Sets)

	Country_Region	WeekOfYear	ConfirmedDaily	FatalitiesDaily
1	NULL	4	878	25
2	NULL	5	10598	217
3	NULL	6	25079	547
4	NULL	7	31908	860
5	NULL	8	9540	792
6	NULL	9	7386	482
7	NULL	10	19502	601
8	NULL	11	50585	2278
9	NULL	12	148281	7154
10	NULL	13	356230	17675
11	NULL	14	536684	33953
12	NULL	NULL	1196671	64584
13	Afghanistan	NULL	299	7
14	Albania	NULL	333	20
15	Algeria	NULL	1251	130
16	Andorra	NULL	466	17
17	Angola	NULL	10	2
18	Antigua and B...	NULL	15	0
19	Argentina	NULL	1451	43
20	Armenia	NULL	770	7
21	Australia	NULL	5550	30
22	Austria	NULL	11781	186
23	Azerbaijan	NULL	521	5

Task 5 : RANK() and DENSE_RANK()

The basic idea of the RANK() function is to assign a rank to each row within a partition of a result set. It adds the number of tied rows to the tied rank to calculate the next rank; hence the ranks assigned may not be consecutive.

select WeekOfYear , Country_Region, ConfirmedDaily , Rank() Over(partition by WeekOfYear order by ConfirmedDaily desc) as Country_Rank from COVID_19_aggr order by WeekOfYear

select WeekOfYear , Country_Region, FatalitiesDaily , Rank() Over(partition by WeekOfYear order by FatalitiesDaily desc) as Country_Rank from COVID_19_aggr order by WeekOfYear

The DENSE_RANK() function works in a similar manner as the RANK() function. The difference is that the DENSE_RANK() function returns consecutive rank values.

select WeekOfYear , Country_Region, FatalitiesDaily , Dense_rank() Over(partition by WeekOfYear order by FatalitiesDaily desc) as Country_Rank from COVID_19_aggr order by WeekOfYear

select WeekOfYear , Country_Region, ConfirmedDaily , Dense_Rank() Over(partition by WeekOfYear order by ConfirmedDaily desc) as Country_Rank from COVID_19_aggr order by WeekOfYear

Sample Result - Task 5 (Rank and Dense Rank)

Rank

	WeekOfYear	Country_Region	ConfirmedDaily	Country_Rank
1	4	China	858	1
2	4	Thailand	5	2
3	4	Singapore	3	3
4	4	Malaysia	3	3
5	4	France	3	3
6	4	Taiwan*	2	6
7	4	Vietnam	2	6
8	4	Korea, South	1	8
9	4	Nepal	1	8
10	4	Bahamas	0	10
11	4	Bhutan	0	10
12	4	Burkina Faso	0	10
13	4	Grenada	0	10
14	4	Cambodia	0	10
15	4	Gabon	0	10
16	4	Iran	0	10
17	4	Luxembourg	0	10

Dense Rank

	WeekOfYear	Country_Region	ConfirmedDaily	Country_Rank
1	4	China	858	1
2	4	Thailand	5	2
3	4	Singapore	3	3
4	4	Malaysia	3	3
5	4	France	3	3
6	4	Taiwan*	2	4
7	4	Vietnam	2	4
8	4	Korea, South	1	5
9	4	Nepal	1	5
10	4	Bahamas	0	6
11	4	Bhutan	0	6
12	4	Burkina Faso	0	6
13	4	Grenada	0	6
14	4	Cambodia	0	6
15	4	Gabon	0	6
16	4	Iran	0	6
17	4	Luxembourg	0	6

Task 5: PERCENT_RANK() and CUME_DIST()

The basic idea of the PERCENT_RANK() function is to evaluate the relative standing of a value within a partition of a result set. It utilizes the PARTITION BY, which distributes the rows into multiple partitions, and ORDER BY clauses, which specifies the logic order of the rows in each partition. The return value of this function always lies between 0 and 1.

```
select  WeekOfYear , Country_Region, ConfirmedDaily
, Format(Percent_Rank() Over(partition by
WeekOfYear order by  ConfirmedDaily ),'P' ) as
Country_Rank from COVID_19_aggr order by
WeekOfYear , ConfirmedDaily desc
```

The basic idea of the CUME_DIST() function is similar to the PERCENT_RANK() function. The difference is that this function calculates the relative position of a value in group of values by calculating the cumulative distribution of that value.

```
select  WeekOfYear , Country_Region, ConfirmedDaily
, Cume_dist() Over(partition by WeekOfYear order by
ConfirmedDaily desc ) as Country_Rank from
COVID_19_aggr order by  WeekOfYear
```

```
select  WeekOfYear , Country_Region, ConfirmedDaily
,Cume_dist() Over(partition by WeekOfYear order by
ConfirmedDaily ) as Country_Rank from
COVID_19_aggr order by  WeekOfYear ,
ConfirmedDaily desc
```

Sample Result - Task 5 (Percent_Rank and Cume_Dist)

Percent Rank

	WeekOfYear	Country_Region	ConfirmedDaily	Country_Rank
1	4	China	858	0.00%
2	4	Thailand	5	0.56%
3	4	Singapore	3	1.12%
4	4	Malaysia	3	1.12%
5	4	France	3	1.12%
6	4	Taiwan*	2	2.79%
7	4	Vietnam	2	2.79%
8	4	Korea, South	1	3.91%
9	4	Nepal	1	3.91%
10	4	Bahamas	0	5.03%
11	4	Bhutan	0	5.03%
12	4	Burkina Faso	0	5.03%
13	4	Grenada	0	5.03%
14	4	Cambodia	0	5.03%
15	4	Gabon	0	5.03%
16	4	Iran	0	5.03%
17	4	Luxembourg	0	5.03%

Cume_Dist()

	WeekOfYear	Country_Region	ConfirmedDaily	Country_Rank
1	4	China	858	0.00555555555555556
2	4	Thailand	5	0.0111111111111111
3	4	Singapore	3	0.0277777777777778
4	4	Malaysia	3	0.0277777777777778
5	4	France	3	0.0277777777777778
6	4	Taiwan*	2	0.0388888888888889
7	4	Vietnam	2	0.0388888888888889
8	4	Korea, South	1	0.05
9	4	Nepal	1	0.05
10	4	Bahamas	0	1
11	4	Bhutan	0	1
12	4	Burkina Faso	0	1
13	4	Grenada	0	1
14	4	Cambodia	0	1
15	4	Gabon	0	1
16	4	Iran	0	1
17	4	Luxembourg	0	1
18	4	Kosovo	0	1

Task6 : PIVOT Tables

The task was to create two PIVOT tables:

PIVOT #1:
Dimension 1: WeekOfYear
Dimension 2: Top 10 Country_Region
Measurement 1: ConfirmedDaily

PIVOT #2:
Dimension 1: WeekOfYear
Dimension 2: Top 10 Country_Region
Measurement 2: FatalitiesDaily

The SQL Server PIVOT operator is used to turn the unique values in one column into multiple columns in the output, and performs aggregations on any remaining column values.

The steps used to create a PIVOT table are:

- Select a base dataset.
- Create a temporary result using a derived table
- Apply the PIVOT operator

PIVOT Table 1

	Countries	4	5	6	7	8	9	10	11	12	13	14
1	1	858	10485	24923	31599	8588	2717	5230	15274	32421	95912	187364
2	2	5	18	61	224	405	2355	4755	6906	22756	47861	52933
3	3	3	13	17	39	349	1066	3891	5891	18983	38894	52743
4	4	3	12	13	18	79	565	1414	3786	17628	35482	38397
5	5	3	12	12	6	59	119	859	3537	9967	23642	32160
6	6	2	10	8	6	28	88	720	2658	7881	14798	25164
7	7	2	8	7	4	13	71	455	1091	5216	12245	20335
8	8	1	7	7	3	8	63	250	1045	3923	7501	16532
9	9	1	5	5	3	5	45	220	943	2678	6732	9297
10	10	0	4	5	1	2	43	184	937	2159	6319	7402

	Countries	4	5	6	7	8	9	10	11	12	13	14
1	1	China	China	China	China	China	Korea, South	Iran	Italy	Italy	US	US
2	2	Thailand	Japan	Diamond Princess	Diamond Princess	Korea, South	China	Italy	Iran	US	Spain	Spain
3	3	Singapore	Singapore	Singapore	Singapore	Diamond Princess	Italy	Korea, South	Spain	Spain	Italy	France
4	4	Malaysia	Thailand	Thailand	Japan	Japan	Iran	China	Germany	Germany	Germany	Germany
5	5	France	Australia	Korea, South	United Kingdom	Italy	Japan	France	France	France	France	Italy
6	6	Taiwan*	Korea, South	Malaysia	Malaysia	Iran	France	Germany	US	Iran	Iran	United Kingdom
7	7	Vietnam	Germany	Taiwan*	Korea, South	Singapore	Diamond Princess	Spain	Switzerland	Switzerland	United Kingdom	Iran
8	8	Korea, South	Taiwan*	Vietnam	Vietnam	Taiwan*	Germany	Switzerland	Korea, South	United Kingdom	Switzerland	Turkey
9	9	Nepal	Malaysia	Germany	Germany	United Arab Emirates	Kuwait	Japan	Norway	Netherlands	Turkey	Belgium
10	10	Bahamas	United Arab Emirates	Japan	Thailand	Thailand	Spain	United Kingdom	United Kingdom	Austria	Belgium	Canada

PIVOT Table 2

	Countries	4	5	6	7	8	9	10	11	12	13	14
1	1	25	217	546	858	780	394	235	1208	3384	5198	6380
2	2	0	0	1	1	5	38	204	466	1180	4607	5965
3	3	0	0	0	1	2	27	102	185	945	1754	5339
4	4	0	0	0	0	2	14	28	121	472	1716	5257
5	5	0	0	0	0	2	4	10	80	253	961	3299
6	6	0	0	0	0	1	4	9	54	213	787	1016
7	7	0	0	0	0	0	1	4	28	125	503	1011
8	8	0	0	0	0	0	0	2	19	75	349	935
9	9	0	0	0	0	0	0	2	16	66	286	930
10	10	0	0	0	0	0	0	1	12	63	189	402

	Countries	4	5	6	7	8	9	10	11	12	13	14
1	1	China	China	China	China	China	China	China	Italy	Italy	Italy	US
2	2	Iraq	Suriname	Philippines	France	Iran	Iran	Italy	Iran	Spain	Spain	Spain
3	3	Guatemala	Uganda	Tanzania	Japan	Diamond Princess	Italy	Iran	Spain	Iran	France	Italy
4	4	Burma	Brunei	Germany	Burundi	Italy	Korea, South	Korea, South	China	France	US	France
5	5	Lebanon	Hungary	Denmark	Malta	Korea, South	Diamond Princess	Spain	France	US	Iran	United Kingdom
6	6	Armenia	Pakistan	Antigua and Barbuda	Belarus	Taiwan*	Japan	France	US	United Kingdom	United Kingdom	Netherlands
7	7	Kuwait	Namibia	Japan	Slovenia	Morocco	France	Iraq	Korea, South	Netherlands	Netherlands	Germany
8	8	Oman	Slovakia	Angola	Thailand	Chad	Switzerland	United Kingdom	United Kingdom	Germany	Germany	Iran
9	9	Ghana	Belgium	Croatia	Holy See	Chile	MS Zaandam	Australia	Japan	China	Belgium	Belgium
10	10	Tanzania	Iraq	Papua New Guinea	Philippines	Haiti	Luxembourg	Japan	Switzerland	Belgium	Switzerland	Switzerland

Task 7: Drill up, Drill down , Slicing, Dicing

Drill Up- is used in tasks involving subtotals.It creates subtotals at any level of aggregation needed, from the most detailed up to grand total i.e climbing up a concept hierarchy for the dimension such as time or geography.

Example:- A query could involve a ROLLUP of year >month>day or country>state>city.

Drill Down-This is a reverse of the ROLL UP operation discussed above. The data is aggregated from a higher level summary to a lower level summary/detailed data.

Slicing:-A slice in a multidimensional array is a column of data corresponding to a single value for one or more members of the dimension. It helps the user to visualize and gather the information specific to a dimension.For example, if a user wanted to know the total number of products sold across all of the market locations (Europe,North America, South America)the user would perform a horizontal slice.

Dicing:-In case of Slicing filtering is done to focus on a particular attribute. Dicing, on the other hand , is more of a zoom feature that selects a subset over all the dimensions, but for specific value of the dimension.

1. Roll-Up <-> Roll-down

Query:

*select Countries , sum(piv.[4]) as Week4 ,sum(piv.[5]) as Week5,sum(piv.[6]) as Week6 ,sum(piv.[7]) as Week7 ,sum(piv.[8]) as Week8 ,sum(piv.[9]) as Week9 ,sum(piv.[10]) as Week10 ,sum(piv.[11]) as Week11 ,sum(piv.[12]) as Week12 ,sum(piv.[13]) as Week13 ,sum(piv.[14]) as Week14 from (select * from (select WeekOfYear,ConfirmedDaily , T.row as "Countries" from (select WeekOfYear,Country_Region , ConfirmedDaily , row_number() over(partition by WeekOfYear order by ConfirmedDaily desc) as row from COVID_19_aggr) as T where T.row <=10) as m Pivot (sum(ConfirmedDaily) FOR m.WeekOfYear IN ([4], [5], [6],[7],[8],[9],[10],[11],[12],[13],[14])) AS pivot_table) as piv group by rollup (Countries)*

	Countries	Week4	Week5	Week6	Week7	Week8	Week9	Week10	Week11	Week12	Week13	Week14
1	1	858	10485	24923	31599	8588	2717	5230	15274	32421	95912	187364
2	2	5	18	61	224	405	2355	4755	6906	22756	47861	52933
3	3	3	13	17	39	349	1066	3891	5891	18983	38894	52743
4	4	3	12	13	18	79	565	1414	3786	17628	35482	38397
5	5	3	12	12	6	59	119	859	3537	9967	23642	32160
6	6	2	10	8	6	28	88	720	2658	7881	14798	25164
7	7	2	8	7	4	13	71	455	1091	5216	12245	20335
8	8	1	7	7	3	8	63	250	1045	3923	7501	16532
9	9	1	5	5	3	5	45	220	943	2678	6732	9297
10	10	0	4	5	1	2	43	184	937	2159	6319	7402
11	NULL	878	10574	25058	31903	9536	7132	17978	42068	123612	289386	442327

2. Slicing Query

```
Create procedure slicing (@dim bit, @number int)

as

if @dim =1

begin

select  * from (select * from (select WeekOfYear,ConfirmedDaily , T.row as "Countries" from (select WeekOfYear,Country_Region , ConfirmedDaily , row_number() over( partition by WeekOfYear order by  ConfirmedDaily desc ) as row from COVID_19_aggr) as T where T.row <=10 ) as m  Pivot (  sum(ConfirmedDaily) FOR m.WeekOfYear IN ( [4], [5], [6],[7],[8],[9],[10],[11],[12],[13],[14] ) ) AS pivot_table) as piv where Countries = @number

end
```

	Countries	4	5	6	7	8	9	10	11	12	13	14
1	7	2	8	7	4	13	71	455	1091	5216	12245	20335

```
begin
declare @temp int
set @temp=@number+3


select  Case @temp When 4 then [4] When 5 then [5] When 6 then [6] When 7 then [7] When 8 then [8] When 9 then [9] When 10 then [10] When 11 then [11] When 12 then [12] When 13 then [13] When 14 then [14] else NULL End as "Week" from (select * from (select WeekOfYear,ConfirmedDaily , T.row as "Countries"  from (select WeekOfYear,Country_Region , ConfirmedDaily , row_number() over( partition by WeekOfYear order by ConfirmedDaily desc ) as row from COVID_19_aggr) as T where T.row <=10 ) as m  Pivot (  sum(ConfirmedDaily) FOR m.WeekOfYear IN ( [4], [5], [6],[7],[8],[9],[10],[11],[12],[13],[14] ) ) AS pivot_table) as piv

End
```

	Week
1	187364
2	52933
3	52743
4	38397
5	32160
6	25164
7	20335
8	16532
9	9297
10	7402

Results

The execution commands used to run the query is:

exec dbo.slicing @dim=0,@number=11

	Week
1	187364
2	52933
3	52743
4	38397
5	32160
6	25164
7	20335
8	16532
9	9297
10	7402

exec dbo.slicing @dim=1,@number=7

	Countries	4	5	6	7	8	9	10	11	12	13	14
1	7	2	8	7	4	13	71	455	1091	5216	12245	20335

3. Dicing Query:

```
Create procedure dicing (@week bit, @country int) as

declare @temp int

set @temp=@week+3

select  Case @temp When 4 then [4] When 5 then [5] When 6 then [6] When 7 then [7] When 8 then [8] When 9 then [9] When 10 then [10] When 11 then [11] When 12 then [12] When 13 then [13] When 14 then [14] else NULL End as "Week" from (select * from (select WeekOfYear,ConfirmedDaily , T.row as "Countries"  from (select WeekOfYear,Country_Region , ConfirmedDaily , row_number() over( partition by WeekOfYear order by  ConfirmedDaily desc ) as row from COVID_19_aggr) as T where T.row <=10 ) as m  Pivot ( sum(ConfirmedDaily) FOR m.WeekOfYear IN ( [4], [5], [6],[7],[8],[9],[10],[11],[12],[13],[14] ) ) AS pivot_table) as piv where Countries=@country

Go
```

Result

The execution command used to run the query is:

```
Exec dbo.dicing @week=1,@country=5
```

	Week
1	3

Thank You