```
-- DAwSQL - Workshop-2 Notes

-- WINDOW FUNCTIONS, PARTITION BY, ORDER BY, AGGREGATE WINDOW FUNCTIONS, WINDOW FRAMES

******* WINDOW FUNCTIONS "OVER" İLE KULLANILIR ******

* AGG() + OVER() AS
* AGG() + OVER(OREDER BY.....) AS
* AGG() + OVER(PARTITION BY.....) AS
* AGG() + OVER(PARTITION BY..... OREDER BY.....) AS

--  OVER()'dan sonra ORDER BY yazmasak da default + 1 önceki değere göre işlem yapar.

select product_id, model_year, sum(list_price) as total_price
from product.product
group by product_id, list_price, model_year
```

	product_id	model_year	total_price
1	1	2018	379.99
2	2	2018	749.99
3	3	2018	999.99
4	4	2018	2899.99
5	5	2018	1320.99
6	6	2018	469.99
7	7	2018	3999.99
8	8	2018	1799.99
9	9	2018	2999.99
10	10	2018	1549.00

```
****** GROUP BY İLE SATIR BAZINDA GÖSTERİLEMEYENLER WINDOW FUNCTIONS İLE GÖSTERİLEBİLİR *****
```

Aşağıdaki tabloda her bir ürün karşısına GROUP BY'dan farklı olarak o ürünün toplam ve yıllık toplam satış tutarları, kümülatif satış toplamları ile her bir satırın bir önceki ve bir sonraki satışla beraber toplamını ayrı ayrı yazdırabildik.

```
select product_id, model_year, list_price,
sum(list_price) over() as total_price,
sum(list_price) over(partition by model_year) as price_by_year,
sum(list_price) over(partition by model_year order by product_id) as cumulative,
sum(list_price) over(partition by model_year order by product_id rows between 1 preceding and
1 following) as window
from product.product;
```

1.SUM 2.SUM 3.SUM 4.SUM

	product_id	model_year	list_price	total	price_by_year	cumulative	window
1	1	2018	379.99	488109.84	25487.78	379.99	1129.98
2	2	2018	749.99	488109.84	25487.78	1129.98	2129.97
3	3	2018	999.99	488109.84	25487.78	2129.97	4649.97
4	4	2018	2899.99	488109.84	25487.78	5029.96	5220.97
5	5	2018	1320.99	488109.84	25487.78	6350.95	4690.97
6	6	2018	469.99	488109.84	25487.78	6820.94	5790.97
7	7	2018	3999.99	488109.84	25487.78	10820.93	6269.97
8	8	2018	1799.99	488109.84	25487.78	12620.92	8799.97
9	9	2018	2999.99	488109.84	25487.78	15620.91	6348.98

```
--------------------------
-- LEAD, LAG, FIRST_VALUE, LAST_VALUE, NTH_VALUE
-- ORDERER BY kullanmak zorunlu, PARTITION BY kullanmak optional
-- First_value, Last_value

select list_price,
first_value(list_price) over(order by list_price) as first_value_by_listprice,
-- first_value(list_price) over(partition by model_year order by list_price) as
first_value_by_modelyear,
last_value(list_price) over(order by list_price) as last_price_by_listprice,
first_value(list_price) over(order by list_price rows between 3 preceding and 3 following)
first_by_3window,
last_value(list_price) over(order by list_price rows between 3 preceding and 3 following) as
last_by_3window
from product.product;
```

	list_price	first_value_by_listprice	last_price_by_listprice	first_by_3window	last_by_3window
1	89.99	89.99	89.99	89.99	149.99
2	109.99	89.99	109.99	89.99	159.99
3	149.99	89.99	149.99	89.99	189.99
4	149.99	89.99	149.99	89.99	189.99
5	159.99	89.99	159.99	109.99	199.99
6	189.99	89.99	189.99	149.99	199.99
7	189.99	89.99	189.99	149.99	209.99
8	199.99	89.99	199.99	159.99	209.99
9	199.99	89.99	199.99	189.99	209.99
10	209.99	89.99	209.99	189.99	209.99
11	209.99	89.99	209.99	199.99	209.99
12	209.99	89.99	209.99	199.99	209.99
13	209.99	89.99	209.99	209.99	229.99
14	209.99	89.99	209.99	209.99	229.99
15	209.99	89.99	209.99	209.99	249.99
16	229.99	89.99	229.99	209.99	249.99
17	229.99	89.99	229.99	209.99	249.99

```
select list_price
from product.product
where model_year = 2018
order by list_price;
```

	list_price
1	269.99
2	269.99
3	269.99
4	269.99
5	299.99
6	379.99
7	429.00
8	449.00
9	449.00
10	469.99
11	499.99
12	529.99
13	549.99
14	549.99
15	599.99
16	599.99
17	599.99
18	749.99
19	999.99
20	1320.99
21	1549.00
22	1680.99
23	1799.99
24	2899.99
25	2999.99
26	3999.99

```sql
select list_price,
first_value(list_price) over(partition by model_year order by list_price) as
first_value_by_modelyear,
last_value(list_price) over(partition by model_year order by list_price) as
last_price_by_listprice,
first_value(list_price) over(partition by model_year order by list_price rows between 3
preceding and 3 following) first_by_3window,
last_value(list_price) over(partition by model_year order by list_price rows between 3
preceding and 3 following) as last_by_3window
from product.product;
```

	list_price	first_value_by_modelyear	last_price_by_listprice	first_by_3window	last_by_3window
1	269.99	269.99	269.99	269.99	269.99
2	269.99	269.99	269.99	269.99	299.99
3	269.99	269.99	269.99	269.99	379.99
4	269.99	269.99	269.99	269.99	429.00
5	299.99	269.99	299.99	269.99	449.00
6	379.99	269.99	379.99	269.99	449.00
7	429.00	269.99	429.00	269.99	469.99
8	449.00	269.99	449.00	299.99	499.99
9	449.00	269.99	449.00	379.99	529.99
10	469.99	269.99	469.99	429.00	549.99
11	499.99	269.99	499.99	449.00	549.99
12	529.99	269.99	529.99	449.00	599.99
13	549.99	269.99	549.99	469.99	599.99
14	549.99	269.99	549.99	499.99	599.99
15	599.99	269.99	599.99	529.99	749.99
16	599.99	269.99	599.99	549.99	999.99
17	599.99	269.99	599.99	549.99	1320.99
18	749.99	269.99	749.99	599.99	1549.00
19	999.99	269.99	999.99	599.99	1680.99
20	1320.99	269.99	1320.99	599.99	1799.99

```sql
-- LEAD(SONRAKİ SATIRLARA), LAG(ÖNCEKİ SATIRLARA)  default:1
select product_id, list_price,
sum(list_price) over() as total,
sum(list_price) over(order by product_id) as cumulative,
lag(list_price, 2) over(order by product_id) as iki_oncesi,
lead(list_price, 3) over(order by product_id) as bir_sonrasi,
sum(list_price) over(order by product_id rows between 1 preceding and 1 following) as uclu
from product.product
order by product_id;
```

	product_id	list_price	total	cumulative	iki_oncesi	bir_sonrasi	uclu
1	1	379.99	488109.84	379.99	NULL	2899.99	1129.98
2	2	749.99	488109.84	1129.98	NULL	1320.99	2129.97
3	3	999.99	488109.84	2129.97	379.99	469.99	4649.97
4	4	2899.99	488109.84	5029.96	749.99	3999.99	5220.97
5	5	1320.99	488109.84	6350.95	999.99	1799.99	4690.97
6	6	469.99	488109.84	6820.94	2899.99	2999.99	5790.97
7	7	3999.99	488109.84	10820.93	1320.99	1549.00	6269.97
8	8	1799.99	488109.84	12620.92	469.99	1680.99	8799.97
9	9	2999.99	488109.84	15620.91	3999.99	549.99	6348.98
10	10	1549.00	488109.84	17169.91	1799.99	269.99	6229.98
11	11	1680.99	488109.84	18850.90	2999.99	269.99	3779.98
12	12	549.99	488109.84	19400.89	1549.00	529.99	2500.97
13	13	269.99	488109.84	19670.88	1680.99	599.99	1089.97
14	14	269.99	488109.84	19940.87	549.99	429.00	1069.97
15	15	529.99	488109.84	20470.86	269.99	449.00	1399.97
16	16	599.99	488109.84	21070.85	269.99	449.00	1558.98
17	17	429.00	488109.84	21499.85	529.99	599.99	1477.99
18	18	449.00	488109.84	21948.85	599.99	269.99	1327.00

```
-- ROW_NUMBER, RANK, DENSE_RANK, (SIRA NUMARASI VER)
-- RANK(AYNI DEĞERLERİN HEPSİNE İLKİNİN İNDEKS NUMARASINI VERİRKEN; SAYAÇ ARKADA ÇALIŞIR)
-- DENSE_RANK(SAYAÇ KALDIĞI YERDEN DEVAM EDER)
-- ORDER BY yapılan sütuna göre sıralandırır
-- ORDER BY kullanmak zorunlu, PARTITION BY kullanmak optional

select product_id, model_year, list_price,
row_number() over(order by product_id) as row_1,
row_number() over(partition by model_year order by product_id) as row_2
from product.product
```

	product_id	model_year	list_price	row_1	row_2
18	18	2018	449.00	18	18
19	19	2018	449.00	19	19
20	20	2018	599.99	20	20
21	21	2018	269.99	21	21
22	22	2018	269.99	22	22
23	23	2018	299.99	23	23
24	24	2018	549.99	24	24
25	25	2018	499.99	25	25
26	26	2018	599.99	26	26
27	27	2019	999.99	27	1
28	28	2019	2499.99	28	2
29	29	2019	999.99	29	3
30	30	2019	999.99	30	4
31	31	2019	1632.99	31	5
32	32	2019	469.99	32	6
33	33	2019	469.99	33	7
34	34	2019	469.99	34	8
35	35	2019	832.99	35	9
36	36	2019	832.99	36	10

```
select model_year, list_price,
rank() over(order by list_price) as ranked,
dense_rank() over(order by list_price) as dense_ranked
from product.product
```

	model_year	list_price	ranked	dense_ranked
1	2020	89.99	1	1
2	2019	109.99	2	2
3	2019	149.99	3	3
4	2019	149.99	3	3
5	2020	159.99	5	4
6	2019	189.99	6	5
7	2019	189.99	6	5
8	2020	199.99	8	6
9	2020	199.99	8	6
10	2020	209.99	10	7
11	2020	209.99	10	7
12	2019	209.99	10	7
13	2019	209.99	10	7
14	2019	209.99	10	7
15	2019	209.99	10	7
16	2020	229.99	16	8
17	2020	229.99	16	8
18	2020	249.99	18	9
19	2020	249.99	18	9

```
-- CUME_DIST, PERCENT_RANK, NTILE
-- CUME_DIST(TOPLAM SATIR SAYISINA BÖLER)
-- PERCENT_RANK(TOPLAM SATIR SAYISI-1'E BÖLER)
-- NTILE(INT) : EŞİT GRUPLARA BÖLER

select product_id,
cume_dist() over(order by product_id) as cume_disted,
percent_rank() over(order by product_id) as p_ranked
from product.product
```

	product_id	cume_disted	p_ranked
1	1	0.00311526479750779	0
2	2	0.00623052959501558	0.003125
3	3	0.00934579439252336	0.00625
4	4	0.0124610591900312	0.009375
5	5	0.0155763239875389	0.0125
6	6	0.0186915887850467	0.015625
7	7	0.0218068535825545	0.01875
8	8	0.0249221183800623	0.021875
9	9	0.0280373831775701	0.025
10	10	0.0311526479750779	0.028125

```
select product_id, list_price, model_year,
ntile(10) over(partition by model_year order by list_price desc) as tiled
from product.product
```

	product_id	list_price	model_year	tiled
1	7	3999.99	2018	1
2	9	2999.99	2018	1
3	4	2899.99	2018	1
4	8	1799.99	2018	2
5	11	1680.99	2018	2
6	10	1549.00	2018	2
7	5	1320.99	2018	3
8	3	999.99	2018	3
9	2	749.99	2018	3
10	16	599.99	2018	4
11	20	599.99	2018	4
12	26	599.99	2018	4
13	24	549.99	2018	5
14	12	549.99	2018	5
15	15	529.99	2018	5

```
-- Question-1
-- Find the weekly order count for the city of San Angelo for the last 52 weeks, and also the
cumulative total. Desired output: [week_number, order_count, cuml_order_count]

select distinct datename(week, o.order_date) as week_number, -- order_date'ten hafta no al
count(o.order_id) over(partition by datename(week, o.order_date)) as total, -- sütun ekler
count(o.order_id) over(order by datename(week, o.order_date)) as weekly_cumulative --sütun
--!!!! toplam kümülatif sayım için ikinci kez "partition by" yazmamamız gerekir.!!!
from sale.orders o join sale.store ss on o.store_id=ss.store_id
where ss.city = 'San Angelo' and
o.order_date between dateadd(year, -1, (select max(order_date) from sale.orders)) and (select
max(order_date) from sale.orders);
```

	week_number	total	weekly_cumulative
1	1	2	2
2	10	3	5
3	12	3	8
4	13	1	9
5	14	1	10
6	15	3	13
7	16	4	17
8	17	2	19
9	18	3	22
10	2	1	23
11	27	1	24
12	28	1	25
13	3	1	26
14	4	1	27
15	48	1	28
16	5	2	30
17	53	1	31
18	6	1	32
19	7	1	33

```
-- !!! Haftaları integer olarak sayıp sıraya sokabilmek için CAST ile integer'a çevirdik.
select distinct cast(datename(week, o.order_date) as integer) as week_number,
count(o.order_id) over(partition by cast(datename(week, o.order_date) as integer)) as total,
count(o.order_id) over(order by cast(datename(week, o.order_date) as integer)) as
weekly_cumulative
from sale.orders o join sale.store ss on o.store_id=ss.store_id
where ss.city = 'San Angelo' and
o.order_date between dateadd(year, -1, (select max(order_date) from sale.orders)) and (select
max(order_date) from sale.orders);
```

	week_number	total	weekly_cumulative
1	1	2	2
2	2	1	3
3	3	1	4
4	4	1	5
5	5	2	7
6	6	1	8
7	7	1	9
8	9	1	10
9	10	3	13
10	12	3	16
11	13	1	17
12	14	1	18
13	15	3	21
14	16	4	25
15	17	2	27

****** WINDOW FRAME İÇİNDE WINDOW FRAME ******

```
-- Question-2 -- Calculate 7-day moving average of the number of products sold between '2018-
03-12' and '2018-04-12'
```

--!!! Her satırda aşağı inen / harektli (hep 7 günü gösteren) bir window frame oluşturmamız isteniyor. "Rows between" ile bunu sağlıyoruz. Order By'ı default bıraksaydık (rows.. yazmasaydık) kümülatif olarak sıralamaya devam ederdi.

-- Sorguyu oluşturma sırası [köşeli parantez içinde gösterilmiştir]

**** !!! Aşağıdaki avg() işlemini sum()'ın olduğu window frame içine yazarsak hata verir.

```sql
select t.order_date, t.sum_q,                                          --[6]
-- avg(t.sum_q) over (order by t.order_date) as moving,
avg(t.sum_q) over(order by t.order_date rows between 6 preceding and current row)as moving [7]
from                                                                   --[5]
    (select distinct o.order_date,                                     --[3]
    sum(i.quantity) over(partition by o.order_date) as sum_q --günlük satış miktarı  [4]
    from sale.orders o join sale.order_item i on o.order_id=i.order_id   --[1]
    where o.order_date between '2018-03-12' and '2018-04-12') as t       --[2]
```

	order_date	sum_q	moving
1	2018-03-12	8	8
2	2018-03-14	19	13
3	2018-03-15	2	9
4	2018-03-16	7	9
5	2018-03-17	4	8
6	2018-03-18	8	8
7	2018-03-19	11	8
8	2018-03-20	13	9
9	2018-03-21	10	7
10	2018-03-23	13	9
11	2018-03-25	1	8
12	2018-03-26	8	9
13	2018-03-27	3	8
14	2018-03-28	15	9
15	2018-03-29	13	9
16	2018-03-30	4	8
17	2018-03-31	1	6

```sql
-- Question-3
-- Identify the bikes by gender and find the total counts and average prices of bikes by
gender.  [gender, count_bikes_by_gender, average_price_by_gender]
-- kadın bisikletlerini tespit için “Women, girl, ladies” kelimelerini arattık.

select distinct product_id, product_name,
case when tt.women_girl = 1 then 'Women' else 'Men' end as gender,
case when tt.women_girl = 1 then sum(tt.women_girl) over() else count(tt.women_girl) over() -
sum(tt.women_girl) over() end as count_bikes_by_gender,
avg(tt.list_price) over(partition by tt.women_girl) as average_price_by_gender
from
    (select product_id, product_name, list_price,
    case when patindex('%irl%', product_name) <> 0 or patindex('%omen%', product_name) <> 0
or patindex('%adies%', product_name) <> 0 then 1 else 0 end as women_girl
    from product.product) as tt
order by product_id
```

	product_id	product_name	gender	count_bikes_by_gender	average_price_by_gender
66	66	Sun Bicycles Revolutions 24 - 2019	Men	241	1650.795186
67	67	Sun Bicycles Revolutions 24 - Girl's - 2019	Women	80	1128.352500
68	68	Sun Bicycles Cruz 3 - 2019	Men	241	1650.795186
69	69	Sun Bicycles Cruz 7 - 2019	Men	241	1650.795186
70	70	Electra Amsterdam Original 3i - 2017/2019	Men	241	1650.795186
71	71	Sun Bicycles Atlas X-Type - 2019	Men	241	1650.795186
72	72	Sun Bicycles Biscayne Tandem 7 - 2019	Men	241	1650.795186
73	73	Sun Bicycles Brickell Tandem 7 - 2019	Men	241	1650.795186
74	74	Electra Cruiser Lux 1 - 2019	Men	241	1650.795186
75	75	Electra Cruiser Lux Fat Tire 1 Ladies - 2019	Women	80	1128.352500
76	76	Electra Girl's Hawaii 1 16" - 2019	Women	80	1128.352500
77	77	Electra Glam Punk 3i Ladies' - 2019	Women	80	1128.352500
78	78	Sun Bicycles Biscayne Tandem CB - 2019	Men	241	1650.795186
79	79	Sun Bicycles Boardwalk (24-inch Wheels) - 2019	Men	241	1650.795186
80	80	Sun Bicycles Brickell Tandem CB - 2019	Men	241	1650.795186
81	81	Electra Amsterdam Fashion 7i Ladies' - 2019	Women	80	1128.352500
82	82	Electra Amsterdam Original 3i Ladies' - 2019	Women	80	1128.352500
83	83	Trek Boy's Kickster - 2017/2019	Men	241	1650.795186
84	84	Sun Bicycles Lil Kitt'n - 2019	Men	241	1650.795186
85	85	Haro Downtown 16 - 2019	Men	241	1650.795186
86	86	Trek Girl's Kickster - 2019	Women	80	1128.352500
87	87	Trek Precaliber 12 Boys - 2019	Men	241	1650.795186

```sql
-- Question-4
-- List the staff information in the ascending order according to their performance.
-- Determine their performance according to how long they waited to receive their next order.
-- [staff_id, first_name, last_name, previous_order, next_order, days_waited, cumulative_days,
average_days]

select t.staff_id, t.first_name, t.last_name, t.previous_order, t.next_order, t.days,
sum(t.days) over (partition by t.staff_id order by(t.next_order)) as cumulative,
avg(t.days) over (partition by t.staff_id) as avg_days
from
    (select st.staff_id, st.first_name, st.last_name, o.order_date as previous_order,
    lead(o.order_date) over(partition by st.staff_id order by o.order_date) as next_order,
    datediff(day, o.order_date, lead(o.order_date) over(partition by st.staff_id order by
o.order_date)) as days
    from sale.staff st join sale.orders o on st.staff_id=o.staff_id) as t
order by avg_days
```

	staff_id	first_name	last_name	previous_order	next_order	days	cumulative	avg_days
10...	7	Neil	Mango	2020-04-22	2020-04-22	0	841	1
10...	7	Neil	Mango	2020-04-22	2020-04-23	1	842	1
10...	7	Neil	Mango	2020-04-23	2020-04-23	0	842	1
10...	7	Neil	Mango	2020-04-23	2020-04-23	0	842	1
10...	7	Neil	Mango	2020-04-23	2020-04-27	4	846	1
10...	7	Neil	Mango	2020-04-27	2020-04-28	1	847	1
10...	7	Neil	Mango	2020-04-28	2020-04-28	0	847	1
10...	7	Neil	Mango	2020-04-28	2020-04-29	1	848	1
10...	7	Neil	Mango	2020-04-29	2020-06-17	49	897	1
10...	7	Neil	Mango	2020-06-17	2020-08-23	67	964	1
10...	7	Neil	Mango	2020-08-23	2020-08-25	2	966	1
10...	7	Neil	Mango	2020-08-25	2020-09-06	12	978	1
10...	2	Maria	Cussona	2020-07-12	NULL	NULL	NULL	5
10...	2	Maria	Cussona	2018-01-01	2018-01-05	4	4	5
10...	2	Maria	Cussona	2018-01-05	2018-01-06	1	5	5
10...	2	Maria	Cussona	2018-01-06	2018-01-14	8	13	5
10...	2	Maria	Cussona	2018-01-14	2018-01-14	0	13	5
10...	2	Maria	Cussona	2018-01-14	2018-01-16	2	15	5
11...	2	Maria	Cussona	2018-01-16	2018-01-16	0	15	5
11...	2	Maria	Cussona	2018-01-16	2018-02-03	18	33	5
11...	2	Maria	Cussona	2018-02-03	2018-02-07	4	37	5
11...	2	Maria	Cussona	2018-02-07	2018-02-12	5	42	5

```sql
----------------------------------
-- Question-5
-- In the street column, clear the string characters that were accidentally added to the end
of the initial numeric expression.

select t1.street,
case when isnumeric(right(t1.subs, 1)) = 0 then substring(t1.subs, 1, len(t1.subs) - 1) else
t1.subs end as [target],
case when isnumeric(right(t1.subs, 1)) = 0 then substring(t1.subs, 1, len(t1.subs) - 1) else
t1.subs end
 + ' ' + substring(street, charindex(' ', street) + 1, len(street)) as new_street_name
from
       (select street, substring(street, 1, charindex(' ', street) - 1) as subs
       from sale.customer) as t1
-- where isnumeric(right(t1.subs, 1)) = 0
```

	street
1	9273 Thorne Ave.
2	910 Vine Street
3	769C Honey Creek St.
4	988 Pearl Lane
5	107 River Dr.
6	769 West Road
7	7014 Manor Station Rd.
8	15 Brown St.
9	8550 Spruce Drive
10	476 Chestnut Ave.
11	8790 Cobblestone Street
12	486 Rock Maple Street

	street	target	new_street_name
1	9273 Thorne Ave.	9273	9273 Thorne Ave.
2	910 Vine Street	910	910 Vine Street
3	769C Honey Creek St.	769	769 Honey Creek St.
4	988 Pearl Lane	988	988 Pearl Lane
5	107 River Dr.	107	107 River Dr.
6	769 West Road	769	769 West Road
7	7014 Manor Station Rd.	7014	7014 Manor Station Rd.
8	15 Brown St.	15	15 Brown St.
9	8550 Spruce Drive	8550	8550 Spruce Drive
10	476 Chestnut Ave.	476	476 Chestnut Ave.
11	8790 Cobblestone Street	8790	8790 Cobblestone Street
12	486 Rock Maple Street	486	486 Rock Maple Street

```
-- *** RECURSIVE CTEs ***

-- Question-6

with recur as
(
    select 1 as num, 1 as num_2  -- bu satır “anchor”: tanımlama yapar ve bir sefer çalışır
    union all                    -- num 10 oluncaya kadar 9 kez (döngüyle) ekleme yapar
    select num + 1, num_2 * 2
    from recur
    where num < 10               -- önce from/where sonra select çalışır
)
select * from recur
```

	num	num_2
1	1	1
2	2	2
3	3	4
4	4	8
5	5	16
6	6	32
7	7	64
8	8	128
9	9	256
10	10	512

```
------

WITH Recursive_CTE AS (

    SELECT 10 AS counter
    UNION ALL
    SELECT counter - 1
    FROM Recursive_CTE
    WHERE counter > 0
)
SELECT REPLICATE('* ', counter)
FROM Recursive_CTE
```

	(No column name)
1	**********
2	*********
3	********
4	*******
5	******
6	*****
7	****
8	***
9	**
10	*
11	

```
WITH Recursive_CTE AS (
    SELECT 1 AS c1, 10 AS c2, 1 AS c3, 10 AS c4
    UNION ALL
    SELECT c1 + 1, c2 - 1, c3 + 1, c4 - 1
    FROM Recursive_CTE
    WHERE c1 < 10
)
SELECT REPLICATE('1', c1), REPLICATE('2', c2), REPLICATE('3', c3), REPLICATE('4', c4),
REPLICATE('1', c1) + '-' + REPLICATE('2', c2) + '-' + REPLICATE('3', c3) + '-' +
REPLICATE('4', c4)
FROM Recursive_CTE
```

	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)
1	1	2222222222	3	4444444444	1-2222222222-3-4444444444
2	11	222222222	33	444444444	11-222222222-33-444444444
3	111	22222222	333	44444444	111-22222222-333-44444444
4	1111	2222222	3333	4444444	1111-2222222-3333-4444444
5	11111	222222	33333	444444	11111-222222-33333-444444
6	111111	22222	333333	44444	111111-22222-333333-44444
7	1111111	2222	3333333	4444	1111111-2222-3333333-4444
8	11111111	222	33333333	444	11111111-222-33333333-444
9	111111111	22	333333333	44	111111111-22-333333333-44
10	1111111111	2	3333333333	4	1111111111-2-3333333333-4

```
--- **** ORTAK SÜTUNU OLMAYAN TABLOLARI BİRLEŞTİRME ****---
-- If we want to select order_ids three by three
with recur as
(
    select 1 as num
    union all
    select num + 3
    from recur
    where num < 100
)
select num, t.order_id, t.order_date, t.shipped_date
from recur join (select * from sale.orders) as t on recur.num=t.order_id
```

	num	order_id	order_date	shipped_date
1	1	1	2018-01-01	2018-01-03
2	4	4	2018-01-03	2018-01-05
3	7	7	2018-01-04	2018-01-05
4	10	10	2018-01-05	2018-01-06
5	13	13	2018-01-08	2018-01-11
6	16	16	2018-01-12	2018-01-15
7	19	19	2018-01-14	2018-01-16
8	22	22	2018-01-16	2018-01-17
9	25	25	2018-01-18	2018-01-21
10	28	28	2018-01-19	2018-01-21
11	31	31	2018-01-20	2018-01-22
12	34	34	2018-01-22	2018-01-23

```
-------------------------------------
-- Question-7
-- List order_id, product_id and list_price of the first order of each day

select t.order_id, t.product_name, t.order_date, t.indexing, t.list_price
from
    (select o.order_id, p.product_name, o.order_date, p.list_price,
    row_number() over(partition by o.order_date order by o.order_id, p.list_price) as
indexing
    from sale.orders o join sale.order_item so on o.order_id=so.order_id join product.product
p on p.product_id=so.product_id) as t
where t.indexing = 1
```

	order_id	product_name	order_date	indexing	list_price
1	1	Electra Townie Original 7D EQ - 2018	2018-01-01	1	599.99
2	3	Electra Townie Original 7D EQ - Women's - 2018	2018-01-02	1	599.99
3	4	Schwinn Suburban DLX Step-Thru - 2018	2018-01-03	1	749.99
4	6	Cannondale Supersix Evo Carbon Ultegra - 2017/2018	2018-01-04	1	449.00
5	9	Trek Slash 8 27.5 - 2018	2018-01-05	1	3999.99
6	12	Surly Straggler 650b - 2018	2018-01-06	1	1680.99
7	13	Electra Cruiser 1 (24-Inch) - 2018	2018-01-08	1	269.99
8	14	Surly Ice Cream Truck Frameset - 2018	2018-01-09	1	469.99
9	16	Electra Cruiser 1 (24-Inch) - 2018	2018-01-12	1	269.99
10	18	Electra Girl's Hawaii 1 (16-inch) - 2017/2018	2018-01-14	1	269.99
11	21	Electra Girl's Hawaii 1 (16-inch) - 2017/2018	2018-01-15	1	269.99
12	22	Electra Girl's Hawaii 1 (16-inch) - 2017/2018	2018-01-16	1	269.99
13	24	Cannondale Supersix Evo Carbon Ultegra - 2017/2018	2018-01-18	1	449.00
14	27	Cannondale Systemsix Ultegra - 2018	2018-01-19	1	449.00
15	29	Electra Cruiser 1 (24-Inch) - 2018	2018-01-20	1	269.99
16	32	Electra Townie Original 7D - 2017/2018	2018-01-21	1	499.99
17	34	Cannondale Synapse Al Disc Sora - 2018	2018-01-22	1	429.00

```
-- Question-8
-- How can we join two tables that do not have shared columns?

select brand_id, brand_name from product.brand
select staff_id, first_name, last_name, email from sale.staff
```

	brand_id	brand_name
1	1	Electra
2	2	Haro
3	3	Redline
4	4	Cannondale
5	5	Schwinn
6	6	Giant
7	7	Sun Bicycles
8	8	Surly
9	9	Trek

	staff_id	first_name	last_name	email
1	1	Fabian	Ellycap	fabian.ellycap@bikes.shop
2	2	Maria	Cussona	maria.cussona@bikes.shop
3	3	Gianna	Setamento	gianna.setamento@bikes.shop
4	4	Billie	Wagon	billie.wagon@bikes.shop
5	5	Jane	Destrey	jane.destrey@bikes.shop
6	6	Elaine	Vince	elaine.vince@bikes.shop
7	7	Neil	Mango	neil.mango@bikes.shop
8	8	Ross	Island	ross.island@bikes.shop
9	9	Alec	Terrell	alec.terrell@bikes.shop
10	10	Charlie	Garden	charlie.garden@bikes.shop

```sql
select *
from
(select brand_id, brand_name, row_number () over(order by brand_id) as index_1 from
product.brand) as T1 join
(select staff_id, first_name, last_name, email, row_number() over(order by staff_id) as
index_2 from sale.staff) as T2 on T1.index_1=T2.index_2
```

	brand_id	brand_name	index_1	staff_id	first_name	last_name	email	index_2
1	1	Electra	1	1	Fabian	Ellycap	fabian.ellycap@bikes.shop	1
2	2	Haro	2	2	Maria	Cussona	maria.cussona@bikes.shop	2
3	3	Redline	3	3	Gianna	Setamento	gianna.setamento@bikes.shop	3
4	4	Cannondale	4	4	Billie	Wagon	billie.wagon@bikes.shop	4
5	5	Schwinn	5	5	Jane	Destrey	jane.destrey@bikes.shop	5
6	6	Giant	6	6	Elaine	Vince	elaine.vince@bikes.shop	6
7	7	Sun Bicycles	7	7	Neil	Mango	neil.mango@bikes.shop	7
8	8	Surly	8	8	Ross	Island	ross.island@bikes.shop	8
9	9	Trek	9	9	Alec	Terrell	alec.terrell@bikes.shop	9

```sql
--------------------------------------
-- Question-9
-- Calculate the stores' weekly cumulative number of orders for 2018

select distinct s.store_id, s.store_name, datepart(week, o.order_date) as week,
count(o.order_id) over (partition by s.store_id, datepart(week, o.order_date)) as weekly_sum,
count(o.order_id) over (partition by s.store_id order by datepart(week, o.order_date)) as
cum_sum
from sale.orders o join sale.store s on o.store_id=s.store_id
where year(o.order_date) = 2018
order by s.store_id, week
```

	store_id	store_name	week	weekly_sum	cum_sum
44	1	Sacramento Bikes	45	1	111
45	1	Sacramento Bikes	46	2	113
46	1	Sacramento Bikes	47	4	117
47	1	Sacramento Bikes	48	3	120
48	1	Sacramento Bikes	49	5	125
49	1	Sacramento Bikes	50	2	127
50	1	Sacramento Bikes	51	1	128
51	1	Sacramento Bikes	52	4	132
52	2	Buffalo Bikes	1	8	8
53	2	Buffalo Bikes	2	2	10
54	2	Buffalo Bikes	3	8	18
55	2	Buffalo Bikes	4	10	28
56	2	Buffalo Bikes	5	8	36
57	2	Buffalo Bikes	6	13	49
58	2	Buffalo Bikes	7	8	57
59	2	Buffalo Bikes	8	7	64
60	2	Buffalo Bikes	9	9	73
61	2	Buffalo Bikes	10	9	82

```sql
--------------------------------------
-- Question-10

-- Write a query that returns both of the followings:
-- - The average product price of orders
-- - Average net amounts of orders
```

```sql
select distinct o.order_id, p.list_price,
avg(p.list_price) over(partition by o.order_id) as avg_price,
avg(p.list_price * oo.quantity * (1-oo.discount)) over () as avg_net_amount
from sale.orders o join sale.order_item oo on o.order_id=oo.order_id
join product.product p on p.product_id=oo.product_id
```

	order_id	list_price	avg_price	avg_net_amount
1	1	599.99	1489.792000	1628.360135
2	1	1549.00	1489.792000	1628.360135
3	1	1799.99	1489.792000	1628.360135
4	1	2899.99	1489.792000	1628.360135
5	2	599.99	599.990000	1628.360135
6	3	599.99	799.990000	1628.360135
7	3	999.99	799.990000	1628.360135
8	4	749.99	749.990000	1628.360135
9	5	429.00	859.330000	1628.360135
10	5	599.99	859.330000	1628.360135
11	5	1549.00	859.330000	1628.360135

```sql
-- Question-11
-- Rearrange the email addresses of customers
-- last_name.name >> if yahoo, change to hotmail; if gmail, change to yahoo; if hotmail,
change to gmail; if otherwise, change to clarusway

select email,
substring(substring(email, 1, charindex('@', email) - 1), 1, charindex('.', substring(email,
1, charindex('@', email) - 1)) - 1) as first,
substring(substring(email, 1, charindex('@', email) - 1), charindex('.', substring(email, 1,
charindex('@', email) - 1)) + 1,
len(substring(email, 1, charindex('@', email) - 1))) as last,
substring(substring(email, 1, charindex('@', email) - 1), charindex('.', substring(email, 1,
charindex('@', email) - 1)) + 1,
len(substring(email, 1, charindex('@', email) - 1))) + '.' +
substring(substring(email, 1, charindex('@', email) - 1), 1, charindex('.', substring(email,
1, charindex('@', email) - 1)) - 1) + '@' +
case
when patindex('%@yahoo%', email) <> 0 then 'hotmail.com'
when patindex('%@gmail%', email) <> 0 then 'yahoo.com'
when patindex('%@hotmail%', email) <> 0 then 'gmail.com'
else 'clarusway.com' end as newmail
from sale.customer
```

	email	first	last	newmail
1	emily.brooks@yahoo.com	emily	brooks	brooks.emily@hotmail.com
2	katie.toodei@yahoo.com	katie	toodei	toodei.katie@hotmail.com
3	tameka.fisher@aol.com	tameka	fisher	fisher.tameka@clarusway.com
4	daryl.spence@aol.com	daryl	spence	spence.daryl@clarusway.com
5	charolette.rice@msn.com	charolette	rice	rice.charolette@clarusway.com
6	lyndsey.bean@hotmail.com	lyndsey	bean	bean.lyndsey@gmail.com
7	latasha.hays@hotmail.com	latasha	hays	hays.latasha@gmail.com
8	jacquline.duncan@yahoo.com	jacquline	duncan	duncan.jacquline@hotmail.com
9	genoveva.baldwin@msn.com	genoveva	baldwin	baldwin.genoveva@clarusway.com
10	pamelia.newman@gmail.com	pamelia	newman	newman.pamelia@yahoo.com

```sql
-- Question-12

-- Is there store-level differences in the shopping attiudes of the customers who drive more
to buy bicycles than drive less
-- [distance, store_state, store_name, total_sum_by_distance, avg_by_store,
total_sum_by_store]

select distinct t.close_to_store, t.store_state, t.store_name,
sum(t.list_price) over(partition by t.close_to_store, t.store_name) as drive_level_sum,
avg(t.list_price) over(partition by t.close_to_store, t.store_name) as drive_level_avg,
sum(t.list_price) over(partition by t.store_name) as store_level
from
(select o.order_id, c.zip_code as zip_home, c.state as home_state, st.zip_code as zip_store,
st.state as store_state, p.list_price, st.store_name,
case when c.zip_code = st.zip_code then 1 else 0 end as close_to_store
from sale.customer c join sale.orders o on c.customer_id=o.customer_id join sale.store st on
st.store_id=o.store_id
join sale.order_item oo on o.order_id=oo.order_id join product.product p on
p.product_id=oo.product_id) as t
order by t.store_state, t.store_name;
```

	close_to_store	store_state	store_name	drive_level_sum	drive_level_avg	store_level
1	0	CA	Sacramento Bikes	1156207.94	1182.216707	1191373.67
2	1	CA	Sacramento Bikes	35165.73	1255.918928	1191373.67
3	0	NY	Buffalo Bikes	3865819.00	1222.586654	3894954.72
4	1	NY	Buffalo Bikes	29135.72	882.900606	3894954.72
5	0	TX	San Angelo Bikes	562822.69	1202.612585	640078.18
6	1	TX	San Angelo Bikes	77255.49	1457.650754	640078.18

```sql
--group by versiyonu
select t.close_to_store, t.store_name, sum(t.list_price), avg(t.list_price)
from
(select o.order_id, c.zip_code as zip_home, c.state as home_state, st.zip_code as zip_store,
st.state as store_state, p.list_price, st.store_name,
case when c.zip_code = st.zip_code then 1 else 0 end as close_to_store
from sale.customer c join sale.orders o on c.customer_id=o.customer_id join sale.store st on
st.store_id=o.store_id
join sale.order_item oo on o.order_id=oo.order_id join product.product p on
p.product_id=oo.product_id) as t
group by t.close_to_store, t.store_name
```

	close_to_store	store_name	(No column name)	(No column name)
1	1	Buffalo Bikes	29135.72	882.900606
2	1	Sacramento Bikes	35165.73	1255.918928
3	0	Buffalo Bikes	3865819.00	1222.586654
4	0	San Angelo Bikes	562822.69	1202.612585
5	0	Sacramento Bikes	1156207.94	1182.216707
6	1	San Angelo Bikes	77255.49	1457.650754