계층형 조회(Connect by)를 사용하는 법을 알아보자!

출처 https://rh-cp.tistory.com/86

0. 테스트 테이블

```
select * from emptest;
```

EMPNO	ENAME	DEPTNO	MGR	JOB	SAL
1000	test1	20	(null)	CLERK	800
1001	test2	30	1000	SALESMAN	1600
1002	test3	30	1000	SALESMAN	1250
1003	test4	20	1000	MANAGER	2975
1004	test5	30	1000	SALESMAN	1250
1005	test6	30	1001	MANAGER	2850
1006	test7	10	1001	MANAGER	2450
1007	test8	20	1006	ANALYST	3000
1008	test9	30	1006	PRESIDENT	5000
1009	test10	30	1002	SALESMAN	1500
1010	test11	20	1002	CLERK	1100
1011	test12	30	1001	CLERK	950
1012	test13	20	1000	ANALYST	3000
1013	test14	10	1000	CLERK	1300

1. 계층 레벨 구하기

```
select max(level)
from emptest
start with mgr is null
connect by prior empno = mgr;
```

MAX(LEVEL)
4

- MAX(level)을 사용하여 트리구조의 최대 깊이를 구합니다.
- START WITH은 시작조건을 의미한다. 즉 mgr이 null인것부터 시작한다는 뜻입니다.
- CONNECT BY PRIOR는 조인조건을 의미한다. 즉 empno와 mgr이 같은 것을 조인하는 것.

2. 계층 구조 조회하기

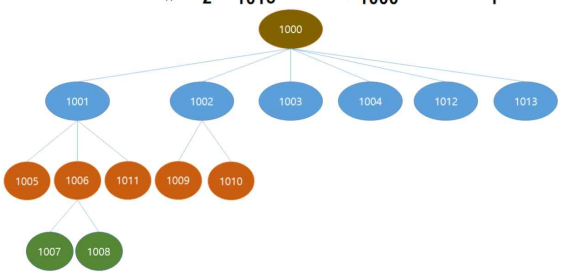
```
select level, empno, deptno, mgr, job, sal
from emptest
start with mgr is null
connect by prior empno = mgr;
```

LEVEL	EMPNO	DEPTNO	MGR	JOB	SAL
1	1000	20	(null)	CLERK	800
2	1001	30	1000	SALESMAN	1600
3	1005	30	1001	MANAGER	2850
4	1006	10	1001	MANAGER	2450
4	1007	20	1006	ANALYST	3000
4	1008	30	1006	PRESIDENT	5000
3	1011	30	1001	CLERK	950
2	1002	30	1000	SALESMAN	1250
3	1009	30	1002	SALESMAN	1500
3	1010	20	1002	CLERK	1100
2	1003	20	1000	MANAGER	2975
2	1004	30	1000	SALESMAN	1250
2	1012	20	1000	ANALYST	3000
2	1013	10	1000	CLERK	1300

- LEVEL을 사용해 계층을 나타내봅시다.
- 계층을 조금 더 명확히 보기 위해 LPAD함수를 사용합니다.
- CONNECT_BY_ISLEAF 는 계층구조에서 가장 최하위(자식이 없는)를 1로 표시해줍니다.

```
select level, lpad(' ', 4 * (level - 1)) || empno, mgr, connect_by_isleaf
from emptest
start with mgr is null
connect by prior empno = mgr;
```

LEVEL	LPAD('',4*(LEVEL-1))||EMPNO	MGR	CONNECT_BY_ISLEAF
1	1000	(null)	0
2	1001	1000	0
3	1005	1001	1
3	1006	1001	0
4	1007	1006	1
4	1008	1006	1
3	1011	1001	1
2	1002	1000	0
3	1009	1002	1
3	1010	1002	1
2	1003	1000	1
2	1004	1000	1
2	1012	1000	1
2	1013	1000	1



3. CONNECT BY 키워드

1. LEVEL : 위에 사용했던것 처럼 검색 항목의 깊이를 의미합니다. 최상위 레벨값은 1입니다.

2. CONNECT_BY_ROOT : 계층구조에서 가장 최상위 값을 표시합니다.

```
select level, empno, ename, mgr, connect_by_root(empno)
from emptest
start with mgr is null
connect by prior empno = mgr;
```

LEVEL	EMPNO	ENAME	MGR	CONNECT_BY_ROOT(EMPNO)
1	1000	test1	(null)	1000
2	1001	test2	1000	1000
3	1005	test6	1001	1000
3	1006	test7	1001	1000
4	1007	test8	1006	1000
4	1008	test9	1006	1000
3	1011	test12	1001	1000
2	1002	test3	1000	1000
3	1009	test10	1002	1000
3	1010	test11	1002	1000
2	1003	test4	1000	1000
2	1004	test5	1000	1000
2	1012	test13	1000	1000
2	1013	test14	1000	1000

3. CONNECT_BY_ISLEAF : 계층구조에서 가장 최하위를 표시합니다. 최하위이면 1 아니면 0 을 나타냅니다 .

4. SYS_CONNECT_BY_PATH : 계층구조의 전체 전개 경로를 표시합니다.

```
select level, lpad(' ', 4 * (level - 1)) || empno, mgr,
substr(sys_connect_by_path(decode(mgr, null, 'root', mgr), mgr), ',', 2)
as sys_connect_by_path
from emptest
start with mgr is null
connect by prior empno = mgr;
```

LEVEL	LPAD('',4*(LEVEL-1))||EMPNO	MGR	SYS_CONNECT_BY_PATH
1	1000	(null)	ROOT
2	1001	1000	ROOT,1000
3	1005	1001	ROOT,1000,1001
3	1006	1001	ROOT,1000,1001
4	1007	1006	ROOT,1000,1001,1006
4	1008	1006	ROOT,1000,1001,1006
3	1011	1001	ROOT,1000,1001
2	1002	1000	ROOT,1000
3	1009	1002	ROOT,1000,1002
3	1010	1002	ROOT,1000,1002
2	1003	1000	ROOT,1000
2	1004	1000	ROOT,1000
2	1012	1000	ROOT,1000
2	1013	1000	ROOT,1000

- SYS_CONNECT_BY_PATH('컬럼', '컬럼 사이에 표시할 문자') 형태로 사용합니다.
- 표시를 깔끔하게 하기 위해 SUBSTR으로 2번째부터 보이게 하였고, DECODE를 사용하여 mgr이 null이면 'ROOT'를 나타나게 하였습니다.

5. NOCYCLE : 순환구조가 발생지점까지만 전개됩니다.

```
update emptest set mgr = 1001 where empno = 1000;

select level, lpad(' ', 4 * (level - 1)) || empno, empno, ename, mgr
from emptest
start with mgr = 1000
connect by prior empno = mgr;
```

```
ORA-01436: CONNECT BY의 루프가 발생되었습니다
01436, 00000 - "CONNECT BY loop in user data"
*Cause:
*Action:
```

- empno 1000번은 최상위인데 mgr를 1001로 바꾸게 된다면 1001의 부모가 1000이 되고 1000의 부모가 1001이 되는 루프가 발생하게 됩니다. 이러한 오류가 발생하게 되었을 때 순환발생지점까지만 나타나게 하는 것이 NOCYCLE입니다.

```
select level, lpad(' ', 4 * (level - 1)) || empno, empno, ename, mgr
from emptest
start with mgr = 1000
connect by nocycle prior empno = mgr;
```

LEVEL	LPAD('',4*(LEVEL-1))||EMPNO	EMPNO	ENAME	MGR
1	1001	1001	test2	1000
2	1000	1000	test1	1001
3	1002	1002	test3	1000
4	1009	1009	test10	1002
4	1010	1010	test11	1002
3	1003	1003	test4	1000
3	1004	1004	test5	1000
3	1012	1012	test13	1000
3	1013	1013	test14	1000
2	1005	1005	test6	1001
2	1006	1006	test7	1001
3	1007	1007	test8	1006
3	1008	1008	test9	1006
2	1011	1011	test12	1001
1	1002	1002	test3	1000
2	1009	1009	test10	1002
2	1010	1010	test11	1002
1	1003	1003	test4	1000
1	1004	1004	test5	1000
1	1012	1012	test13	1000
1	1013	1013	test14	1000

6. CONNECT_BY_ISCYCLE : 순환구조 발생 지점을 표시합니다.

```
select level, lpad(' ', 4 * (level - 1)) || empno, empno, ename, mgr, connect_by_iscycle
from emptest
start with mgr = 1000
connect by nocycle prior empno = mgr;
```

LEVEL	LPAD('',4*(LEVEL-1))||EMPNO	EMPNO	ENAME	MGR	CONNECT_BY_ISCYCLE
1	1001	1001	test2	1000	0
2	1000	1000	test1	1001	1
3	1002	1002	test3	1000	0
4	1009	1009	test10	1002	0
4	1010	1010	test11	1002	0
3	1003	1003	test4	1000	0
3	1004	1004	test5	1000	0
3	1012	1012	test13	1000	0
3	1013	1013	test14	1000	0
2	1005	1005	test6	1001	0
2	1006	1006	test7	1001	0

- 순환구조가 발생한 곳을 1로 표시합니다.
- CONNECT_BY_ISCYCLE은 NOCYCLE과 같이 사용합니다.

7. SIBLINGS : 계층구조에서 상관관계를 유지하며 정렬을 해줍니다.

```
select level, lpad(' ', 4 * (level - 1)) || empno, mgr, ename
from emptest
start with mgr is null
connect by prior empno = mgr
order by ename;
```

- 기존 ORDER BY를 사용하여 정렬하였을 때는 아래와 같이 관계가 깨집니다.

LEVEL	LPAD('',4*(LEVEL-1))||EMPNO	MGR	ENAME
1	1000	(null)	test1
3	1009	1002	test10
3	1010	1002	test11
3	1011	1001	test12
2	1012	1000	test13
2	1013	1000	test14
2	1001	1000	test2
2	1002	1000	test3
2	1003	1000	test4
2	1004	1000	test5
3	1005	1001	test6
3	1006	1001	test7
4	1007	1006	test8
4	1008	1006	test9

- ORDER BY 사이에 SIBLINGS를 넣어 정렬을 하게 되면 아래처럼 관계가 유지되며 정렬됩니다.

```
select level, lpad(' ', 4 * (level - 1)) || empno, mgr, ename
from emptest
start with mgr is null
connect by prior empno = mgr
order siblings by ename;
```

LEVEL	LPAD('',4*(LEVEL-1))||EMPNO	MGR	ENAME
1	1000	(null)	test1
2	1012	1000	test13
2	1013	1000	test14
2	1001	1000	test2
3	1011	1001	test12
3	1005	1001	test6
3	1006	1001	test7
4	1007	1006	test8
4	1008	1006	test9
2	1002	1000	test3
3	1009	1002	test10
3	1010	1002	test11
2	1003	1000	test4
2	1004	1000	test5

윈도우 함수(Window Function)의 소중함을 느껴보자

https://schatz37.tistory.com/12

0. 윈도우 함수(Window Function)란?

= 행과 행간의 관계를 쉽게 정의 하기 위해 만든 함수

기존의 SQL 언어는 컬럼과 컬럼간의 연산, 비교, 집계에 특화되어있는 언어였습니다.
반면 행과 행간의 관계를 정의하거나 비교, 연산하는 것은 하나의 SQL 문으로 처리하기 어려웠죠. 이러한 부분을 쉽게 처리하기 위해 생겨난 것이 윈도우 함수입니다.

윈도우 함수의 생김새를 살펴보면 아래와 같습니다.
함수(컬럼) OVER (Partition by 컬럼 Order by 컬럼)

- 함수 : Min, Max, Sum, Count, Rank 등과 같은 기존의 함수 or 윈도우 함수용으로 추가된 함수 (Row_number 등)
- OVER : over 은 윈도우 함수에서 꼭 들어가야 하며 Over 내부에 Partition By 절과 Order by 절이 들어갑니다.
- partition by : 전체 집합을 어떤 기준(컬럼)에 따라 나눌지를 결정하는 부분.
- order by : 어떤 항목(컬럼)을 기준으로 순위를 정할 지 결정하는 부분.

적용할 함수와 Over 절은 윈도우 함수에서 필수적으로 사용되며, 어떤 결과를 만들어낼지에 따라 partition by 와 order by 절을 사용하게 됩니다.

1. group by 와 차이점

윈도우 함수의 생김새를 살펴보면 어떤 기준에 따라 Partition by, 즉 나누어 집계한다는 것을 알 수 있습니다. 그렇다면 group by 와는 어떤 차이가 있을까요?

	group by	윈도우 함수
기능	자르기 + 집약	자르기
특징	1. group by 구에 지정된 컬럼으로 데이터를 자르고 2. 집계 함수를 이용해 집약시킨다.	1. partition by 구에 지정된 컬럼으로 데이터를 자른다.
차이점	행의 수가 줄어든다	행의 수가 그대로 유지된다.

group by 와 윈도우 함수의 가장 큰 차이는 '집약의 과정이 존재하는가? 입니다.

< group by 를 사용한 경우>

```
select address, count(*)
from address
group by address;
```

	address character varying (32)	count bigint
1	서울시	3
2	부산시	2
3	서귀포시	1
4	속초시	1
5	인천시	2

< 윈도우 함수를 이용한 경우 >

```
select address , count(*) over(partition by address)
from address;
```

	address character varying (32)	count bigint
1	부산시	2
2	부산시	2
3	서귀포시	1
4	서울시	3
5	서울시	3
6	서울시	3
7	속초시	1
8	인천시	2
9	인천시	2

group by 는 집약 기능으로 인해 행 수가 줄어든 반면, 윈도우 함수는 행 수가 그대로 남아있습니다.
윈도우 함수에는 집약의 기능이 없기 때문입니다.
이러한 특징을 이용해서 우리는 행과 행간의 관계를 편하게 다룰 수 있게 됩니다!

2. 레코드에 순번 붙이기

각 행에 순번을 붙이는 경우 윈도우 함수에서는 아주 간단하게 row_number 함수를 사용하면 됩니다. 우선 예제로 사용할 데이터의 생김새는 이렇습니다.

	class [PK] integer	student_id [PK] character (4)	weight integer
1	1	100	50
2	1	101	55
3	1	102	56
4	2	100	60
5	2	101	72
6	2	102	73
7	2	103	73

< 윈도우 함수 사용 시 >

```
select class, student_id , row_number() over(order by class, student_id)
from weights;
```

	class [PK] integer	student_id [PK] character (4)	row_number bigint
1	1	100	1
2	1	101	2
3	1	102	3
4	2	100	4
5	2	101	5
6	2	102	6
7	2	103	7

이렇게 간단하게 행에 대한 순번을 만들 수 있습니다.
그렇다면 row_number 함수를 지원하지 않는 DBMS를 사용하고 있거나 지원하지 않았을 때에는 어떻게 만들었을까요?

< 윈도우 함수를 사용하지 않을 때 >

```
select class, student_id
      , (select count(*) from weights a
         where (a.class, a.student_id) <= (b.class, b.student_id)) as rownum
from weights b;
```

위와 같이 select절에 서브쿼리를 활용하면 윈도우 함수를 사용했을 때와 같은 결과를 얻을 수 있습니다.

3. 누적합 구하기 (class별 student_id 의 몸무게의 누적합 구하기)

누적합을 만들고 싶을 땐 간단하게 윈도우 함수 구문에 sum 함수를 사용해주면 됩니다.

< 윈도우 함수를 사용할 때 >

```
select class, student_id, weight, sum(weight) over(partition by class order by student_id) as cum_weight
from weights;
```

	class [PK] integer	student_id [PK] character (4)	weight integer	cum_weight bigint
1	1	100	50	50
2	1	101	55	105
3	1	102	56	161
4	2	100	60	60
5	2	101	72	132
6	2	102	73	205
7	2	103	73	278

sum 을 윈도우 함수에서 활용할 때, order by절을 사용하면 순차적인 누적합을 구할 수 있습니다. 만약 order by절을 사용하지 않는다면, partition by 로 구분한 파티션에 존재하는 값들의 합이 도출됩니다.
(이 부분은 바로 다음 예시인 그룹 내 비율 구하기에서 활용해보겠습니다.)

< 윈도우 함수를 사용하지 않을 때 >

윈도우 함수를 사용하지 않는 경우의 프로세스를 하나씩 따라가 보도록 하겠습니다.

```
select a.class, a.student_id, a.weight , sum(b.weight) as cum_weight
from weights a
join weights b
on (a.class = b.class and a.student_id >= b.student_id) --이게 하나의 파티션이 됨.
group by 1,2,3
order by 1,2,3
```

4. 정말 윈도우 함수가 성능적으로 우수할까?

이렇게 윈도우 함수를 사용할 때와 그렇지 않을때를 비교해봤습니다.
윈도우 함수를 사용하면 무엇보다 아주 간단하게 쿼리를 작성할 수 있다는 장점이 있습니다.
그리고 테이블 스캔 횟수도 적으니 더 성능적으로 좋다고 생각이 들기도 합니다.

하지만 윈도우 함수는 행과 행간의 관계를 다루는 함수이기 때문에 윈도우 함수를 사용하면 기본적으로 정렬(Sort)의 과정이 생기게 됩니다.
정렬이 발생한다는 말은 SQL 의 성능이 저하된다는 의미입니다.
그렇기 때문에 테이블 스캔 횟수가 적다는 이유로 무조건 윈도우 함수를 사용하는게 좋아!라는 결론은 잘못된 결론이 됩니다.

윈도우 함수를 잘 사용하기 위해서는?
그렇다면 윈도우 함수를 잘 사용하기 위해서는 결국 불필요한 정렬을 줄여야 합니다.
기본적인 접근은 이전의 서브쿼리나 조인에 대한 게시글에서도 살펴봤듯, 스캔을 해야 할 행의 수를 줄이는 것입니다.
이를 위해서는 Join 이나 서브쿼리를 통해 레코드 수를 줄인 후 윈도우 함수를 사용하는 등의 방법을 고려해보는 것이 좋습니다.

해당 부분에 대한 내용도 이번 포스팅에서 살펴보고 싶지만, 아직까지 공부한 부분이 부족해 다음에 따로 포스팅을 해야할 것 같습니다.
우선 윈도우 함수가 성능적으로 좋지 않기 때문에 성능을 생각한다면 사용할 때 주의해야 된다는 점을 기억해두시면 좋겠습니다.

윈도우함수 - 순위함수 (정미나 유투브)

1. 날짜별 팔린 음료 개수의 순위 구하기

0. 기본 테이블 구조

```
SELECT ORDER_DT,
       COUNT(*)
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT
 ORDER BY ORDER_DT;
```

	ORDER_DT	COUNT(*)
1	20190801	8
2	20190802	13
3	20190803	8
4	20190804	14
5	20190805	7
6	20190806	9
7	20190807	12
8	20190808	8
9	20190809	11
10	20190810	10

어떤 날이 음료가 가장 많이 팔렸나?? 순위를 매겨보자.

1. RANK 함수

```
SELECT ORDER_DT,
       COUNT(*),
       RANK() OVER(ORDER BY COUNT(*) DESC) AS RANK
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT
```

	ORDER_DT	COUNT(*)	RANK
1	20190804	14	1
2	20190802	13	2
3	20190807	12	3
4	20190809	11	4
5	20190810	10	5
6	20190806	9	6
7	20190803	8	7
8	20190808	8	7
9	20190801	8	7
10	20190805	7	10

2. DENSE_RANK 함수

```
SELECT ORDER_DT,
       COUNT(*),
       DENSE_RANK() OVER(ORDER BY COUNT(*) DESC) AS RANK
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT
;
```

	ORDER_DT	COUNT(*)	RANK
1	20190804	14	1
2	20190802	13	2
3	20190807	12	3
4	20190809	11	4
5	20190810	10	5
6	20190806	9	6
7	20190803	8	7
8	20190808	8	7
9	20190801	8	7
10	20190805	7	8

3. ROW_NUMBER 함수

```
SELECT ORDER_DT,
       COUNT(*),
       ROW_NUMBER() OVER(ORDER BY COUNT(*) DESC) AS RANK
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT
;
```

	ORDER_DT	COUNT(*)	RANK
1	20190804	14	1
2	20190802	13	2
3	20190807	12	3
4	20190809	11	4
5	20190810	10	5
6	20190806	9	6
7	20190803	8	7
8	20190808	8	8
9	20190801	8	9
10	20190805	7	10

2. 날짜별, 음료별 팔린 음료 개수의 순위 구하기

0. 기본 테이블 구조

```
SELECT ORDER_DT,
       ORDER_ITEM,
       COUNT(*)
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT, ORDER_ITEM
 ORDER BY ORDER_DT;
```

	ORDER_DT	ORDER_ITEM	COUNT(*)
1	20190801	바닐라 프라푸치노	1
2	20190801	아메리카노	3
3	20190801	자바칩 프라푸치노	1
4	20190801	카페라떼	2
5	20190801	쿨라임 피지오	1
6	20190802	그린티크림 프라푸치노	1
7	20190802	아메리카노	7
8	20190802	카페라떼	1
9	20190802	카페모카	1
10	20190802	콜드브루	2
11	20190802	한라봉주스	1
12	20190803	바닐라 프라푸치노	1
13	20190803	아메리카노	3
14	20190803	자바칩 프라푸치노	1
15	20190803	카페라떼	1
16	20190803	쿨라임 피지오	1

어떤 날에 어떤 음료가 가장 많이 팔렸나?? 순위를 매겨보자.

1. RANK 함수

```
SELECT ORDER_DT,
       ORDER_ITEM,
       COUNT(*),
       RANK() OVER(PARTITION BY ORDER_DT ORDER BY COUNT(*) DESC) AS RANK
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT, ORDER_ITEM
 ORDER BY ORDER_DT;
```

	ORDER_DT	ORDER_ITEM	COUNT(*)	RANK
1	20190801	아메리카노	3	1
2	20190801	카페라떼	2	2
3	20190801	자바칩 프라푸치노	1	3
4	20190801	바닐라 프라푸치노	1	4
5	20190801	쿨라임 피지오	1	5
6	20190802	아메리카노	7	1
7	20190802	콜드브루	2	2
8	20190802	그린티크림 프라푸치노	1	3
9	20190802	카페라떼	1	4
10	20190802	카페모카	1	5
11	20190802	한라봉주스	1	6
12	20190803	아메리카노	3	1
13	20190803	바닐라 프라푸치노	1	2
14	20190803	쿨라임 피지오	1	3

날짜별로 1위를 한 음료들만 뽑아보자.

```
SELECT * FROM (
SELECT ORDER_DT,
       ORDER_ITEM,
       COUNT(*),
       ROW_NUMBER() OVER(PARTITION BY ORDER_DT ORDER BY COUNT(*) DESC) AS RANK
  FROM STARBUCKS_ORDER
 GROUP BY ORDER_DT, ORDER_ITEM
) WHERE RANK = 1;
```

	ORDER_DT	ORDER_ITEM	COUNT(*)	RANK
1	20190801	아메리카노	3	1
2	20190802	아메리카노	7	1
3	20190803	아메리카노	3	1
4	20190804	아메리카노	6	1
5	20190805	아메리카노	3	1
6	20190806	아메리카노	4	1
7	20190807	아메리카노	5	1
8	20190808	아메리카노	4	1
9	20190809	아메리카노	4	1
10	20190810	아메리카노	4	1

윈도우 함수 - ROWS , RANGE

출처 http://www.gurubee.net/lecture/2674

ROWS 사용 예제1

아래는 부서별(PARTITION BY deptno)로 이전 ROW(ROWS 1 PRECEDING)의 급여와 현재 ROW의 급여 합계를 출력하는 예제이다

```
SELECT empno, ename, deptno, sal,
       SUM(sal) OVER (PARTITION BY deptno
                      ORDER BY empno
                       ROWS 1 PRECEDING ) pre_sum
  FROM emp;


-- PRE_SUM : 이전 ROW와 현재 ROW의 급여 합계가 출력된 것을 확인 할 수 있다.
 EMPNO ENAME       DEPTNO        SAL    PRE_SUM
------ ------- ---------- ---------- ----------
  7782 CLARK           10       2450       2450
  7839 KING            10       5000       7450
  7934 MILLER          10       1300       6300
  7369 SMITH           20        800        800
  7566 JONES           20       2975       3775
  7788 SCOTT           20       3000       5975
  7876 ADAMS           20       1100       4100
  7902 FORD            20       3000       4100
  7499 ALLEN           30       1600       1600
  7521 WARD            30       1250       2850
  7654 MARTIN          30       1250       2500
  7698 BLAKE           30       2850       4100
  7844 TURNER          30       1500       4350
  7900 JAMES           30        950       2450
```

ROWS 사용 예제2

아래 예제는 첫 번째 ROW부터 마지막 ROW까지의 합과(SAL1), 첫 번째 ROW부터 현재 ROW까지의 합(SAL2) 그리고 현재 ROW부터 마지막 ROW까지의 합(SAL3)을 출력하는 예제이다.

```
SELECT empno, ename, deptno, sal,
       SUM(sal) OVER(ORDER BY deptno, empno
                 ROWS BETWEEN UNBOUNDED PRECEDING
                          AND UNBOUNDED FOLLOWING) sal1,
       SUM(sal) OVER(ORDER BY deptno, empno
                 ROWS BETWEEN UNBOUNDED PRECEDING
                          AND CURRENT ROW) sal2,
       SUM(sal) OVER(ORDER BY deptno, empno
                 ROWS BETWEEN CURRENT ROW
                          AND UNBOUNDED FOLLOWING) sal3
  FROM emp;


-- SAL1 : 첫 번째 ROW부터 마지막 ROW까지의 급여 합계이다.
-- SAL2 : 첫 번째 ROW 부터 현재 ROW까지의 급여 합계이다.
-- SAL3 : 현재 ROW부터 마지막 ROW까지 급여 합계이다.
 EMPNO ENAME       DEPTNO        SAL       SAL1       SAL2       SAL3
------ ------- ---------- ---------- ---------- ---------- ----------
  7782 CLARK           10       2450      29025       2450      29025
  7839 KING            10       5000      29025       7450      26575
  7934 MILLER          10       1300      29025       8750      21575
  7369 SMITH           20        800      29025       9550      20275
  7566 JONES           20       2975      29025      12525      19475
  7788 SCOTT           20       3000      29025      15525      16500
  7876 ADAMS           20       1100      29025      16625      13500
  7902 FORD            20       3000      29025      19625      12400
  7499 ALLEN           30       1600      29025      21225       9400
  7521 WARD            30       1250      29025      22475       7800
  7654 MARTIN          30       1250      29025      23725       6550
  7698 BLAKE           30       2850      29025      26575       5300
  7844 TURNER          30       1500      29025      28075       2450
  7900 JAMES           30        950      29025      29025        950
```

RANGE 사용 예제

아래는 월별 금액 리스트를 출력하고, 직전 3개월 합계(AMT_PRE3)와 이후 3개월 합계(AMT_FOL3)를 함께 표시하는 예제이다.

아래 예제에서는 7월 데이터가 없기 때문에 직전 3개월 합계(AMT_PRE3) 8월의 경우 6월5월 두 달치만 누적된 것을 확인 할 수 있다.

```
WITH test AS
(
SELECT '200801' yyyymm, 100 amt FROM dual
UNION ALL SELECT '200802', 200 FROM dual
UNION ALL SELECT '200803', 300 FROM dual
UNION ALL SELECT '200804', 400 FROM dual
UNION ALL SELECT '200805', 500 FROM dual
UNION ALL SELECT '200806', 600 FROM dual
UNION ALL SELECT '200808', 800 FROM dual
UNION ALL SELECT '200809', 900 FROM dual
UNION ALL SELECT '200810', 100 FROM dual
UNION ALL SELECT '200811', 200 FROM dual
UNION ALL SELECT '200812', 300 FROM dual
)
SELECT yyyymm
     , amt
     , SUM(amt) OVER(ORDER BY TO_DATE(yyyymm,'yyyymm')
                 RANGE BETWEEN INTERVAL '3' MONTH PRECEDING
                           AND INTERVAL '1' MONTH PRECEDING) amt_pre3
     , SUM(amt) OVER(ORDER BY TO_DATE(yyyymm,'yyyymm')
                 RANGE BETWEEN INTERVAL '1' MONTH FOLLOWING
                           AND INTERVAL '3' MONTH FOLLOWING) amt_fol3
  FROM test
;
-- AMT_PRE3 : 직전 3개월 합계
-- AMT_FOL3 : 이후 3개월 합계
YYYYMM            AMT   AMT_PRE3   AMT_FOL3
--------- ---------- ---------- ----------
200801            100                   900
200802            200        100       1200
200803            300        300       1500
200804            400        600       1100
200805            500        900       1400
200806            600       1200       1700
200808            800       1100       1200
200809            900       1400        600
200810            100       1700        500
200811            200       1800        300
200812            300       1200
```

오라클 LAG, LEAD 함수 사용법 (이전값, 다음값)

https://gent.tistory.com/339

오라클에서 이전 행의 값을 찾거나 다음 행의 값을 찾기 위해서는 LAG, LEAD 함수를 사용하면 된다.

LAG(expr [,offset] [,default]) **OVER**([partition_by_clause] order_by_clause)
LEAD(expr [,offset] [,default]) **OVER**([partition_by_clause] order_by_clause)

- LAG 함수 : 이전 행의 값을 리턴
- LEAD 함수 : 다음 행의 값을 리턴

LAG, LEAD 사용 예제1

```
SELECT empno
     , ename
     , job
     , sal
     , LAG(empno) OVER(ORDER BY empno)  AS empno_prev
     , LEAD(empno) OVER(ORDER BY empno) AS empno_next
  FROM emp
 WHERE job IN ('MANAGER', 'ANALYST', 'SALESMAN')
```

EMPNO	ENAME	JOB	SAL	EMPNO_PREV	EMPNO_NEXT
7499	ALLEN	SALESMAN	1600		7521
7521	WARD	SALESMAN	1250	7499	7566
7566	JONES	MANAGER	2975	7521	7654
7654	MARTIN	SALESMAN	1250	7566	7698
7698	BLAKE	MANAGER	2850	7654	7782

LAG, LEAD 사용 예제2

```
SELECT empno
     , ename
     , job
     , sal
     , LAG(empno, 2) OVER(ORDER BY empno) AS empno_prev
  FROM emp
 WHERE job IN ('MANAGER', 'ANALYST', 'SALESMAN')
```

EMPNO	ENAME	JOB	SAL	EMPNO_PREV
7499	ALLEN	SALESMAN	1600	
7521	WARD	SALESMAN	1250	
7566	JONES	MANAGER	2975	7499
7654	MARTIN	SALESMAN	1250	7521
7698	BLAKE	MANAGER	2850	7566

LAG, LEAD 사용 예제3

```
SELECT empno
     , ename
     , job
     , sal
     , LAG(empno, 2, 9999) OVER(ORDER BY empno) AS empno_prev
  FROM emp
 WHERE job IN ('MANAGER', 'ANALYST', 'SALESMAN')
```

EMPNO	ENAME	JOB	SAL	EMPNO_PREV
7499	ALLEN	SALESMAN	1600	9999
7521	WARD	SALESMAN	1250	9999
7566	JONES	MANAGER	2975	7499
7654	MARTIN	SALESMAN	1250	7521

LAG, LEAD 사용 예제4

```
SELECT empno
     , ename
     , job
     , sal
     , LAG(job) OVER(PARTITION BY job ORDER BY empno) AS empno_prev
  FROM emp
 WHERE job IN ('MANAGER', 'ANALYST', 'SALESMAN')
```

EMPNO	ENAME	JOB	SAL	EMPNO_PREV
7788	SCOTT	ANALYST	3000	
7902	FORD	ANALYST	3000	ANALYST
7566	JONES	MANAGER	2975	
7698	BLAKE	MANAGER	2850	MANAGER
7782	CLARK	MANAGER	2450	MANAGER
7499	ALLEN	SALESMAN	1600	
7521	WARD	SALESMAN	1250	SALESMAN
7654	MARTIN	SALESMAN	1250	SALESMAN
7844	TURNER	SALESMAN	1500	SALESMAN