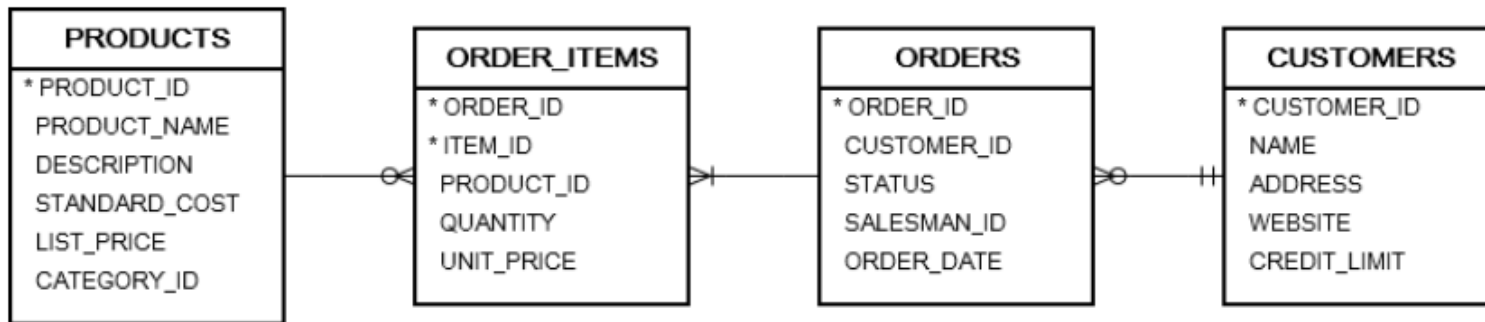


데이터 분석 SQL Fundamentals

Join(조인)

조인

- 관계형 DB에서 가장 기본이자 중요한 기능
- 두개 이상의 테이블을 서로 **연결**하여 데이터를 추출
- 관계형 DB에서는 조인을 통해 서로 다른 테이블간의 정보를 원하는 대로 가져올 수 있음.



조인 - 서로 다른 테이블을 연결


emp 테이블

ename	empno	job	deptno
ALLEN	7,499	SALESMAN	30
BLAKE	7,698	MANAGER	30
CLARK	7,782	MANAGER	10
FORD	7,902	ANALYST	20
JAMES	7,900	CLERK	30
JONES	7,566	MANAGER	20
KING	7,839	PRESIDENT	10
MARTIN	7,654	SALESMAN	30

dept 테이블

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Join
On
emp.deptno = dept.deptno



ename	empno	job	deptno	dname
ALLEN	7,499	SALESMAN	30	SALES
BLAKE	7,698	MANAGER	30	SALES
CLARK	7,782	MANAGER	10	ACCOUNTING
FORD	7,902	ANALYST	20	RESEARCH
JAMES	7,900	CLERK	30	SALES
JONES	7,566	MANAGER	20	RESEARCH
KING	7,839	PRESIDENT	10	ACCOUNTING
MARTIN	7,654	SALESMAN	30	SALES

조인 시 데이터 집합 레벨의 변화

emp 테이블

조인 컬럼 deptno
기준 **M** 집합

ename	empno	job	deptno
ALLEN	7,499	SALESMAN	30
BLAKE	7,698	MANAGER	30
CLARK	7,782	MANAGER	10
FORD	7,902	ANALYST	20
JAMES	7,900	CLERK	30
JONES	7,566	MANAGER	20
KING	7,839	PRESIDENT	10
MARTIN	7,654	SALESMAN	30

조인 컬럼 deptno
기준 **1** 집합

dept 테이블

deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
select a.ename, empno, job, a.deptno, dname
from hr.emp a
join hr.dept b on a.deptno = b.deptno
order by ename
```

조인 결과 **M** 집합

ename	empno	job	deptno	dname
ALLEN	7,499	SALESMAN	30	SALES
BLAKE	7,698	MANAGER	30	SALES
CLARK	7,782	MANAGER	10	ACCOUNTING
FORD	7,902	ANALYST	20	RESEARCH
JAMES	7,900	CLERK	30	SALES
JONES	7,566	MANAGER	20	RESEARCH
KING	7,839	PRESIDENT	10	ACCOUNTING
MARTIN	7,654	SALESMAN	30	SALES

1:M 조인 시 결과 집합은
M집합의 레벨을 그대로 유지

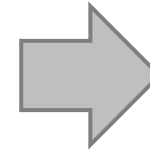
SQL 조인 시 데이터 집합 레벨의 변화 - 1

상품 주문 테이블 M(중복)

사용자 ID	주문 ID	상품 ID
1	D001	A001
1	D001	A002
1	D002	A001
2	D003	A001
2	D003	A002
2	D003	A003
2	D004	A003

1(Unique) 상품 테이블

상품 ID	상품명
A001	치솔
A002	치약
A003	비누



조인 결과

사용자 ID	주문 ID	상품 ID	상품명
1	D001	A001	치솔
1	D001	A002	치약
1	D002	A001	치솔
2	D003	A001	치솔
2	D003	A002	치약
2	D003	A003	비누
2	D004	A003	비누

SQL 조인 시 데이터 집합 레벨의 변화 - 2

1(Unique) 고객 테이블

고객ID	고객명
1	김길동
2	권철민
3	김옥빈

M(중복) 고객 연락처테이블

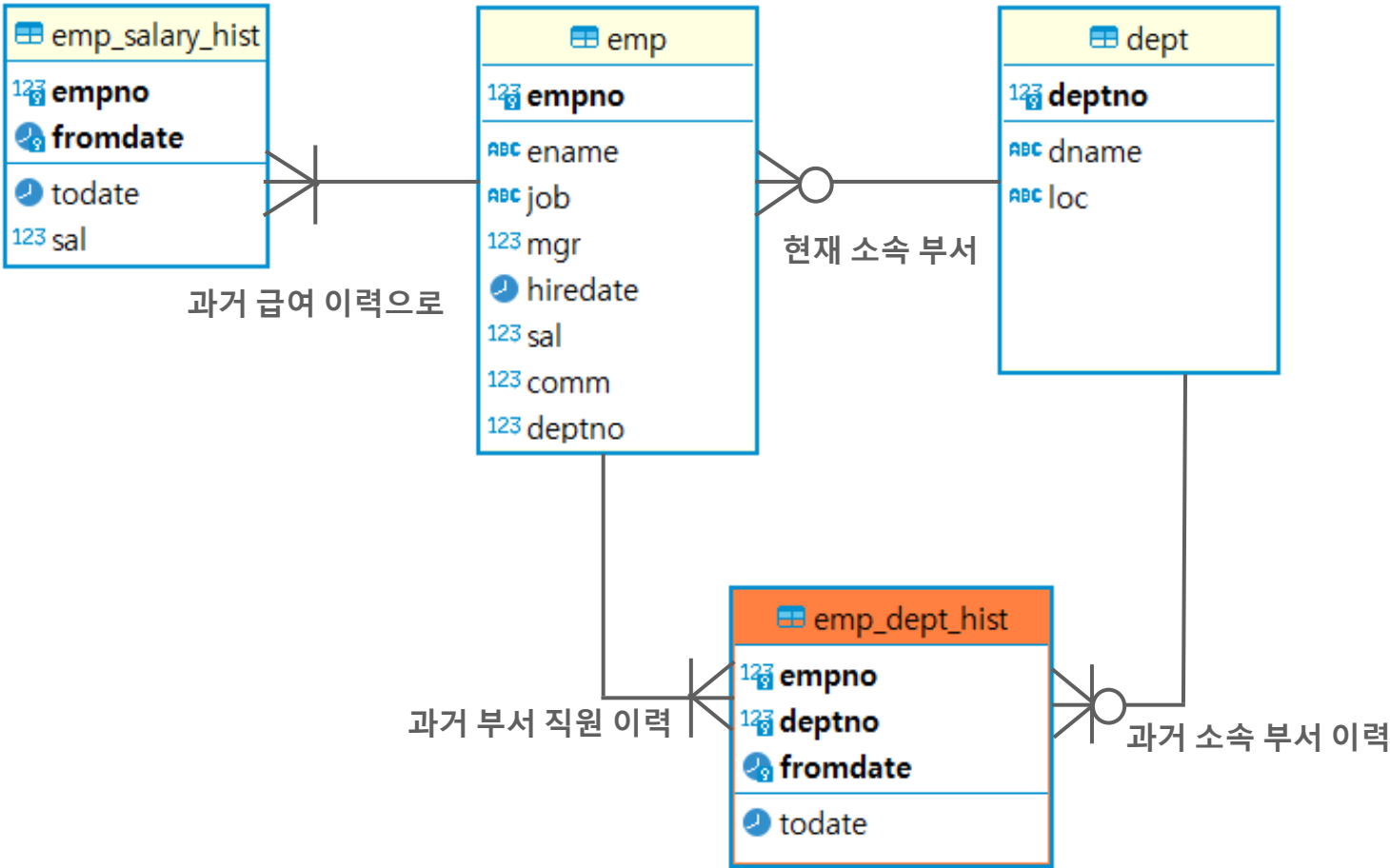
고객ID	구분	연락처
1	전화	010X
1	주소	경기도
1	직장	서울



조인 결과

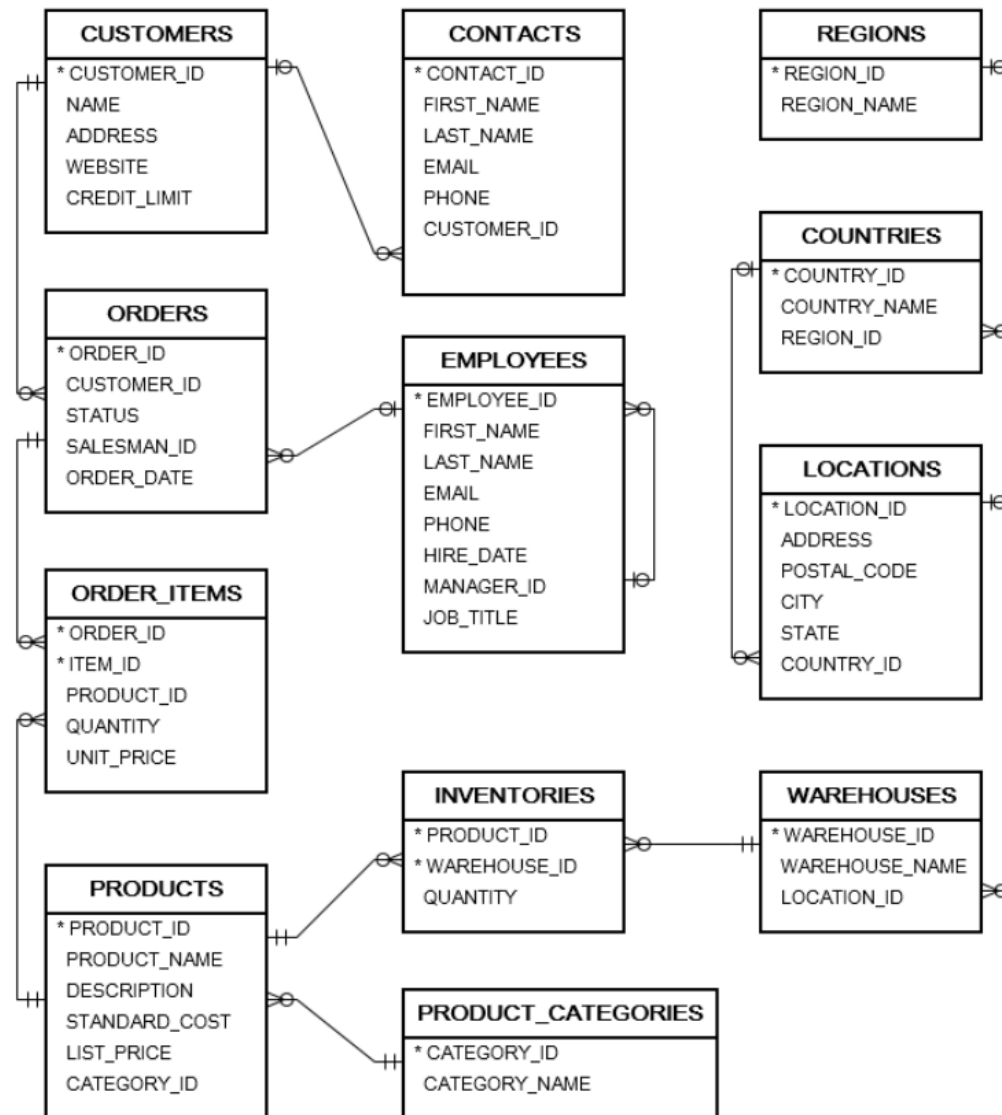
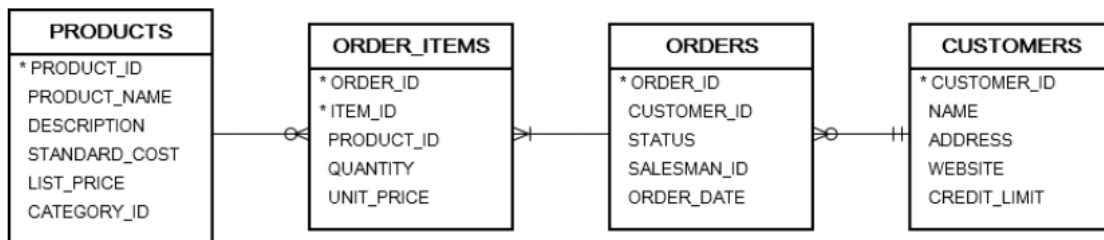
고객ID	구분	고객명	연락처
1	전화	김길동	010X
1	주소	김길동	경기도
1	직장	김길동	서울

HR 스키마의 주요 테이블 관계(ERD)

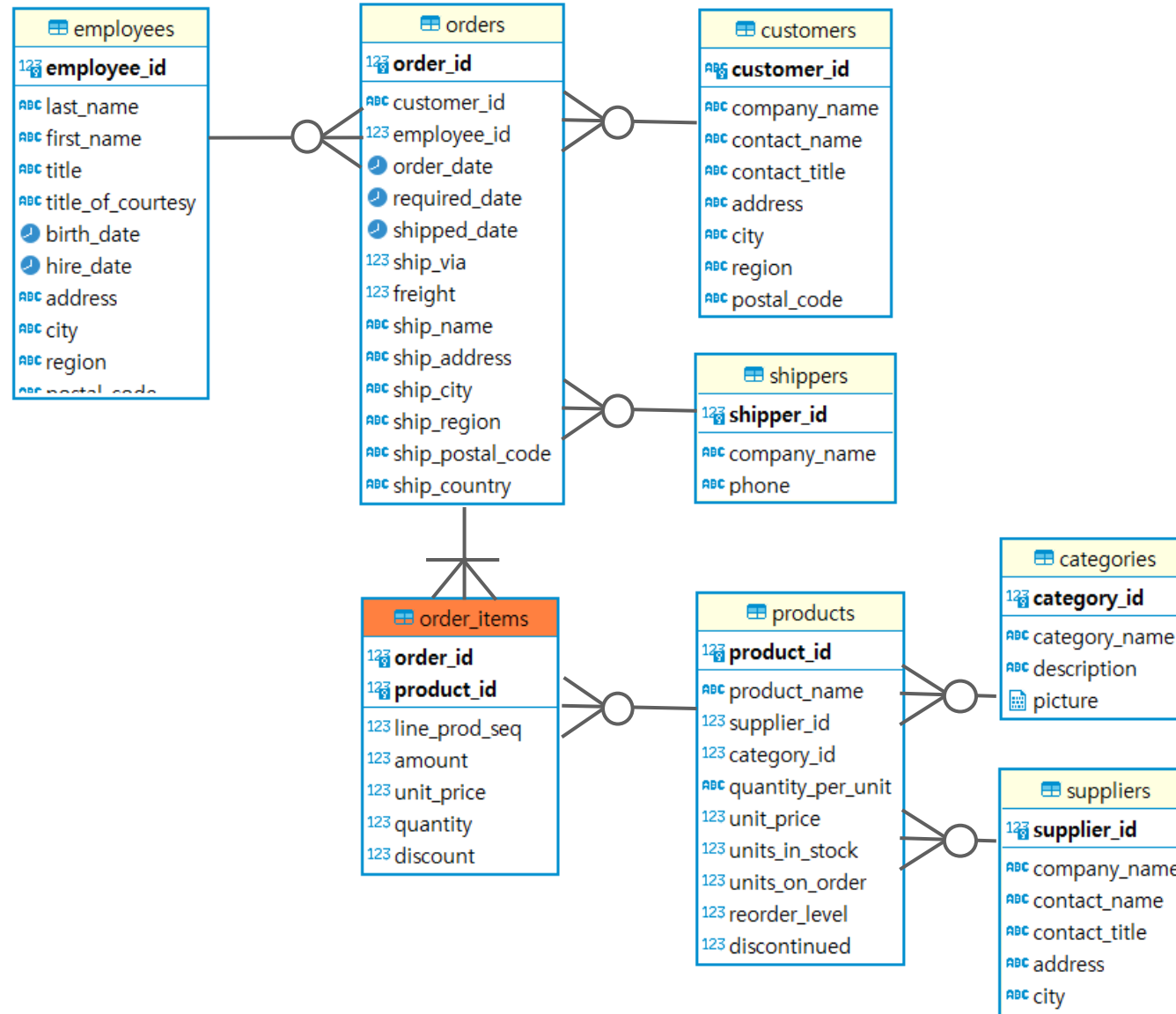


데이터 연결 관계 이해

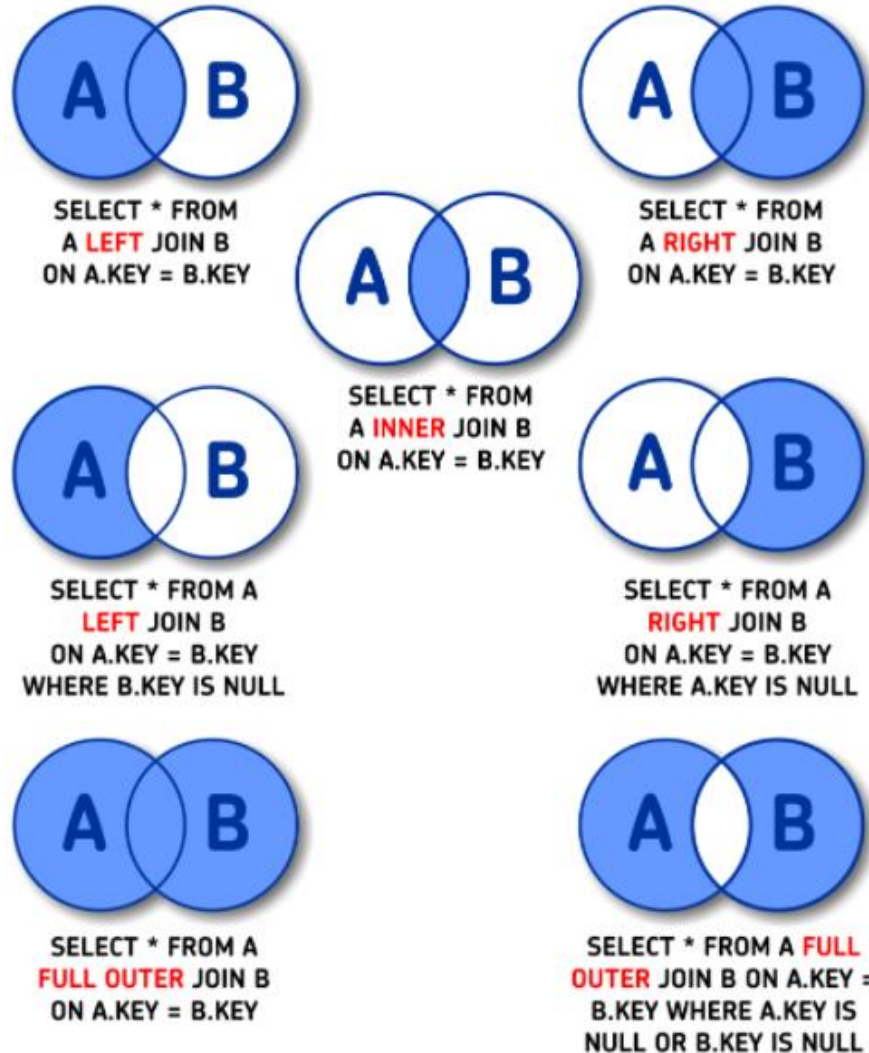
1:1, 1:M, M:N 관계의 이해



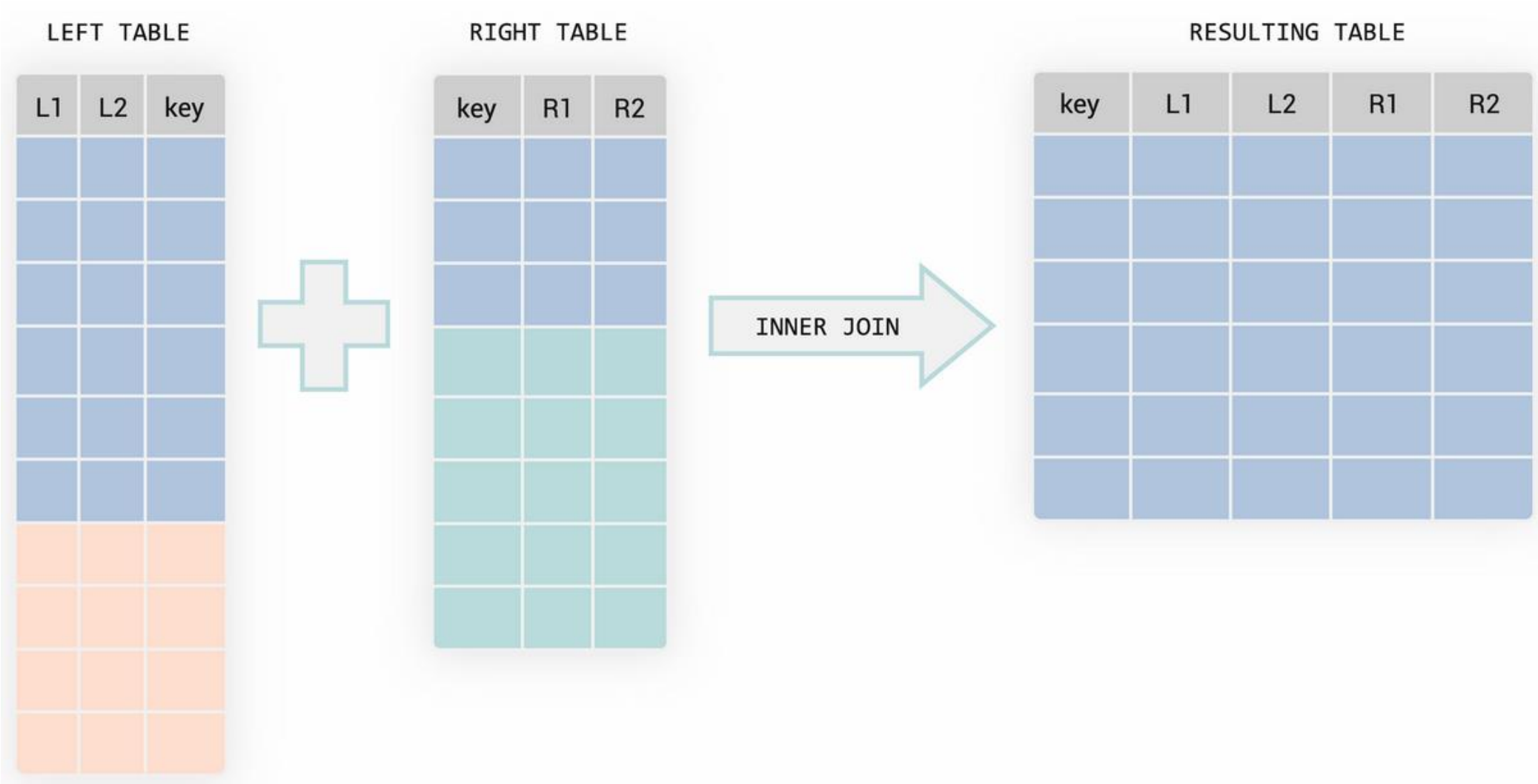
NW 스키마 주요 테이블 관계(ERD)



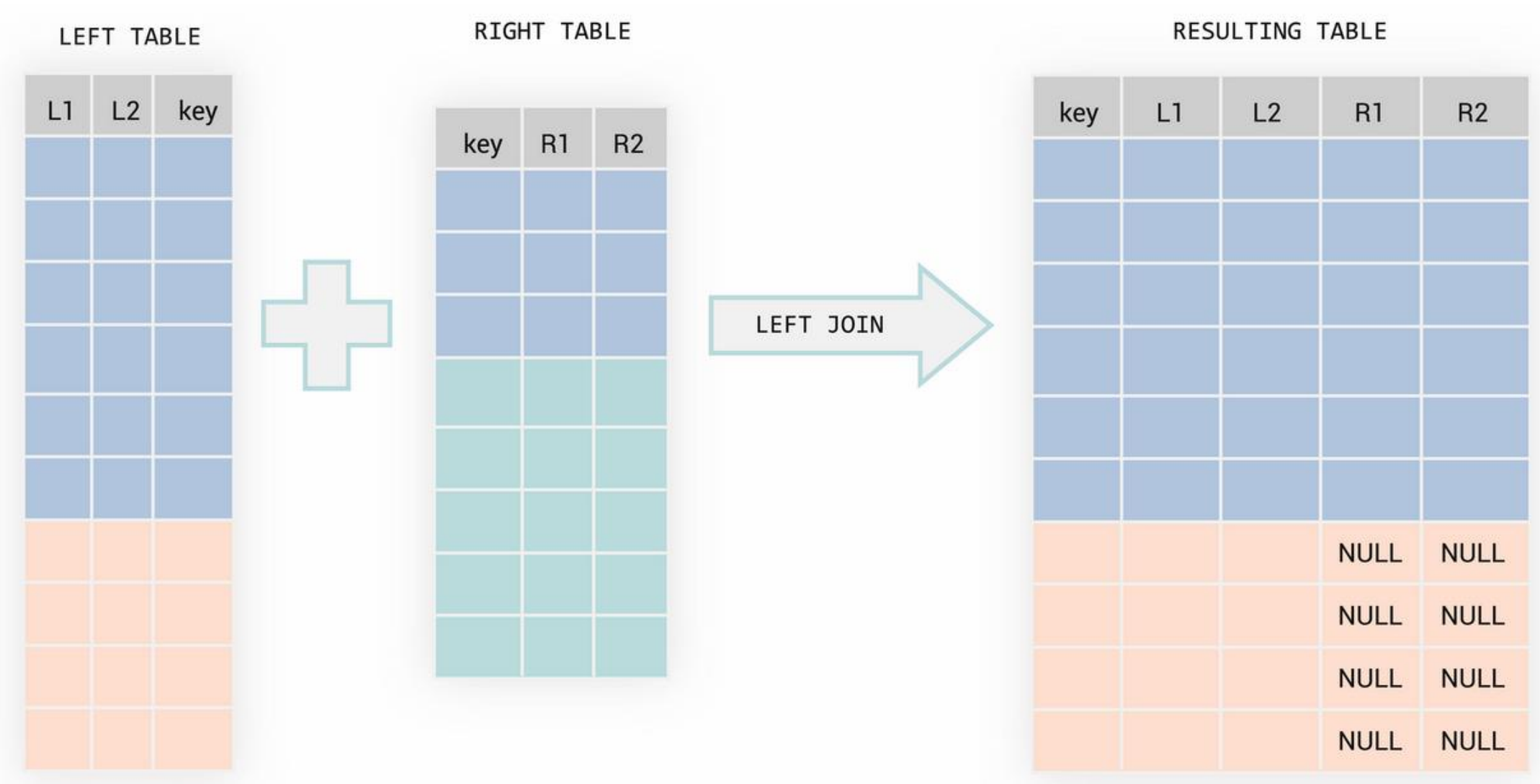
조인 유형 – Inner Join, Outer join, Full Outer Join



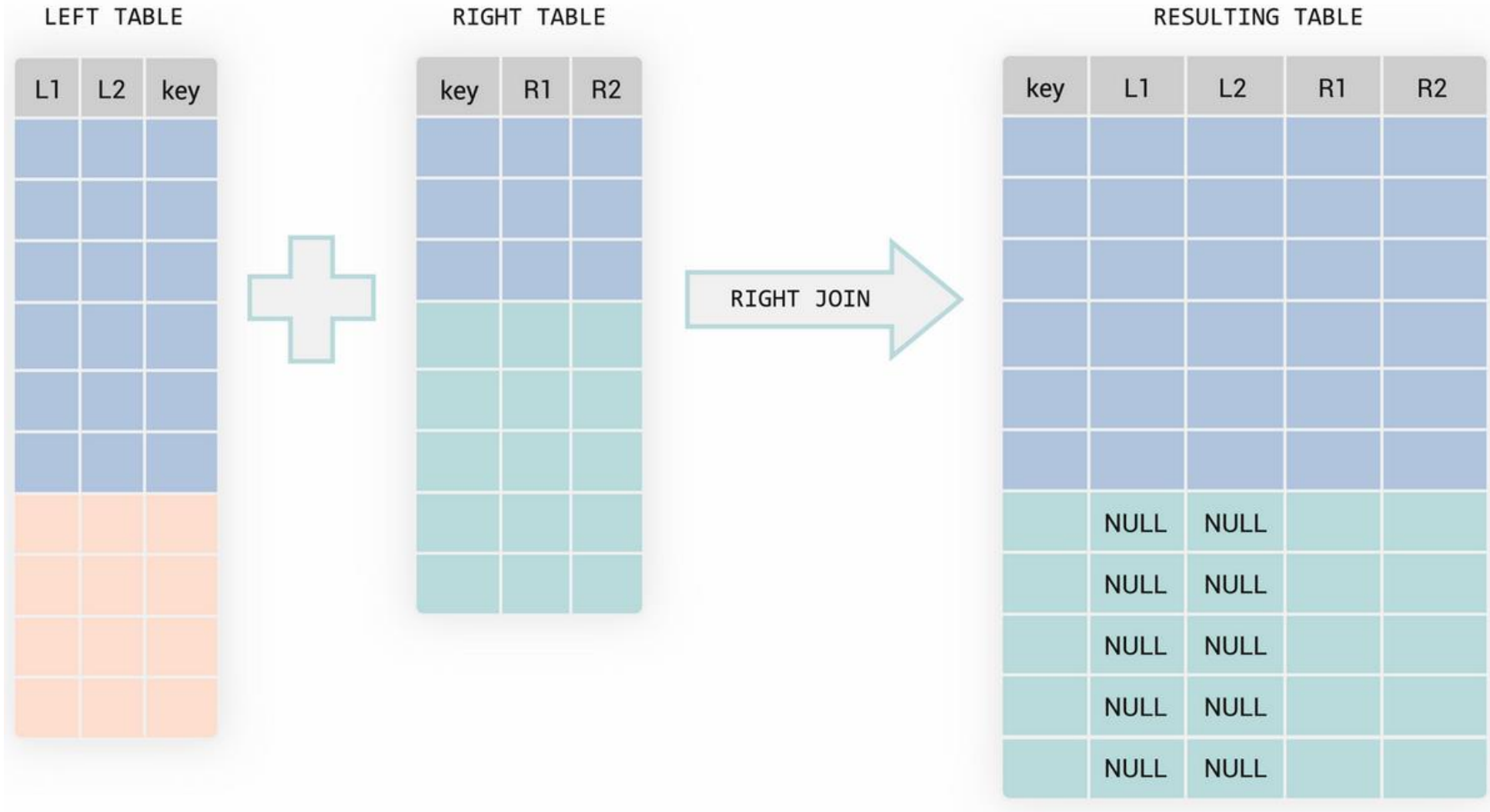
Inner 조인



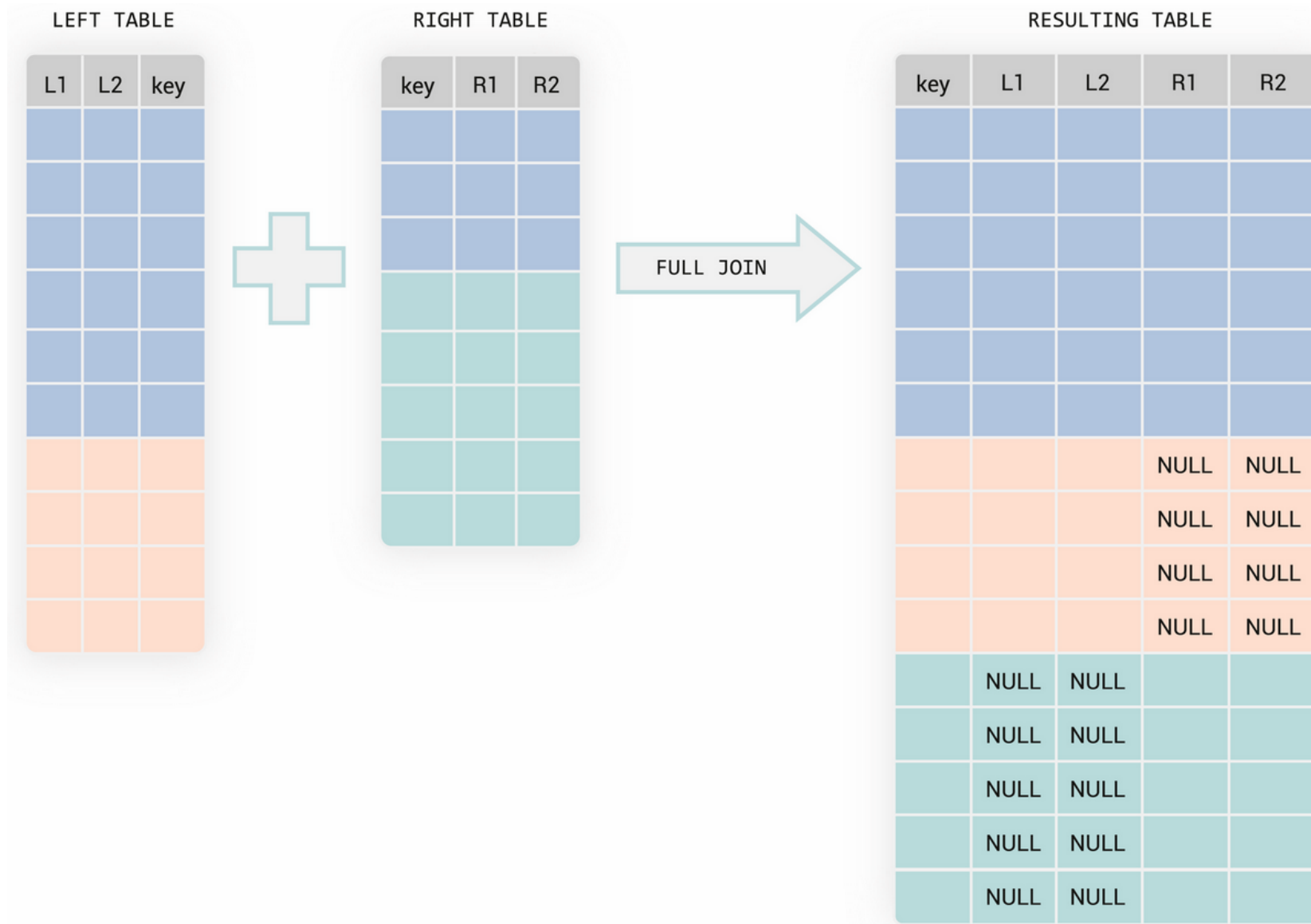
Left (Outer) 조인



Right (Outer) Join



Full Outer 조인



Non Equi 조인

- Equi 조인 : 조인 시 연결하는 키값이 서로 같은 경우(즉 = 로 연결)
- Non Equi 조인: 키 값으로 연결 시 = 이 아닌 다른 연산자(between, >, >=, <, <=)를 사용하는 조인

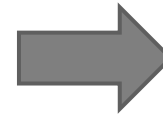
Between 조인 적용

직원정보와 급여등급 정보를 가져오기

EMP		
empno	ename	sal
7369	SMITH	800
7499	ALLEN	1600
7521	WARD	1250
7566	JONES	2975
7654	MARTIN	1250
7698	BLAKE	2850
7782	CLARK	2450
7839	KING	5000
7844	TURNER	1500
7900	JAMES	950
7902	FORD	3000
7934	MILLER	1300

Between
조인

SALGRADE		
grade	losal	hisal
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999



select empno, ename, sal, b.grade as salgrade
from hr.emp a join hr.salgrade b on a.sal
between b.losal and b.hisal;

조인 결과			
empno	ename	sal	salgrade
7369	SMITH	800	1
7900	JAMES	950	1
7521	WARD	1250	2
7654	MARTIN	1250	2
7934	MILLER	1300	2
7499	ALLEN	1600	3
7844	TURNER	1500	3
7566	JONES	2975	4
7698	BLAKE	2850	4
7782	CLARK	2450	4
7902	FORD	3000	4
7839	KING	5000	5

Cross 조인(Cartesian Product 조인)

조인 컬럼없이 두 테이블 간 가능한 모든 연결을 결합하는 조인 방식

```
Select a.*, b.* From table a cross join table b
```

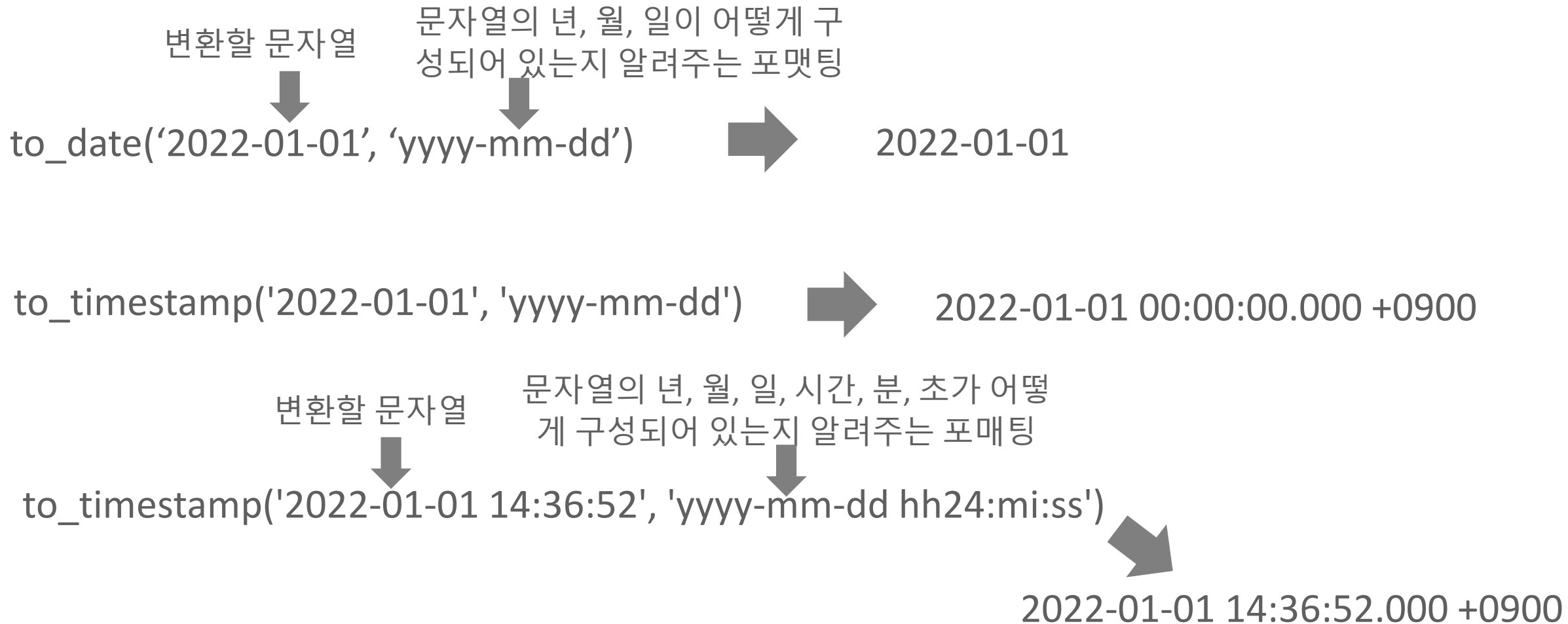
Table a		Table b		조인 결과	
Color	Cross join	Size		Color	Size
Red		Small		Red	Small
Blue		Medium		Blue	Small
		Large		Red	Medium
		Extra Large		Blue	Medium
				Red	Large
				Blue	Large
				Red	Extra Large
				Blue	Extra Large

Date, Timestamp, Interval 다루기

Date/Timestamp/Time/Interval 타입

Date	일자로서 년, 월, 일 정보를 가짐. YYYY-MM-DD
Timestamp	일자를 시간 정보까지 같이 가짐. YYYY-MM-DD HH24:MI:SS
Time	오직 시간 정보만 가짐. HH24:MI:SS
Interval	N days HH24:MI_SS

문자열을 Date, Timestamp로 변환



Date, Timestamp를 문자열로 변환

Date 컬럼 문자열의 년, 월, 일 출력 포매팅

↓ ↓

to_char(hiredate, 'yyyy-mm-dd') ➡ 1980-12-17

날짜, 시간 포매팅 패턴

yyyy-mm-dd hh24:mi:ss

YYYY-MM 내용

am

포매팅 패턴	내용
hh24	하루중 시간(00-23)
<u>hh12</u>	하루중 시간(01-12)
mi	분(00-59)
ss	초(00-59)
yyyy	년도
mm	월(01-12)
dd	일(월중 일자 01-31)

13시
PM 1시

포매팅 패턴	내용
<u>month</u>	월 이름
<u>MONTH</u>	월 이름
<u>day</u>	요일 이름
<u>DAY</u>	요일 이름
<u>w</u>	월의 주(1-5)
<u>ww</u>	년의 주(1-52)
<u>d</u>	요일. <u>일요일</u> (1) ~ <u>토요일</u> (7)
am 또는 pm	AM 또는 PM 표시
<u>tz</u>	시간대

January January
JANUARY

Sunday

1 4
5

d

tz

9:00

::date, ::timestamp, ::text를 이용하여 편리하게 형 변환

- Date를 Timestamp로 변환

```
select to_date('2022-01-01', 'yyyy-mm-dd')::timestamp;
```

- Timestamp를 Text로 변환

```
select to_timestamp('2022-01-01', 'yyyy-mm-dd')::text;
```

- Timestamp를 Date로 변환.

```
select to_timestamp('2022-01-01 14:36:52', 'yyyy-mm-dd hh24:mi:ss')::date
```


extract와 date_part를 이용하여 Date/Timestamp에서 년,월,일/시간,분,초 추출

```
select a.*  
      , extract(year from hiredate) as year  
      , extract(month from hiredate) as month  
      , extract(day from hiredate) as day  
from hr.emp a;
```

```
select a.*  
      , date_part('year', hiredate) as year  
      , date_part('month', hiredate) as month  
      , date_part('day', hiredate) as day  
from hr.emp a;
```

날짜와 시간 연산 - interval의 활용

- ◆ Date 타입에 숫자값을 더하거나/빼면 숫자값에 해당하는 일자를 더하거나/빼서 날짜 계산.

```
select to_date('2022-01-01', 'yyyy-mm-dd') + 2
```

➡ 2022-01-03

- ◆ Timestamp타입에 숫자값을 더하거나 빼면 오류 발생.

```
select to_timestamp('2022-01-01 14:36:52', 'yyyy-mm-dd hh24:mi:ss') + 7
```

➡ 오류 발생

- ◆ Timestamp는 interval 타입을 이용하여 연산 수행.

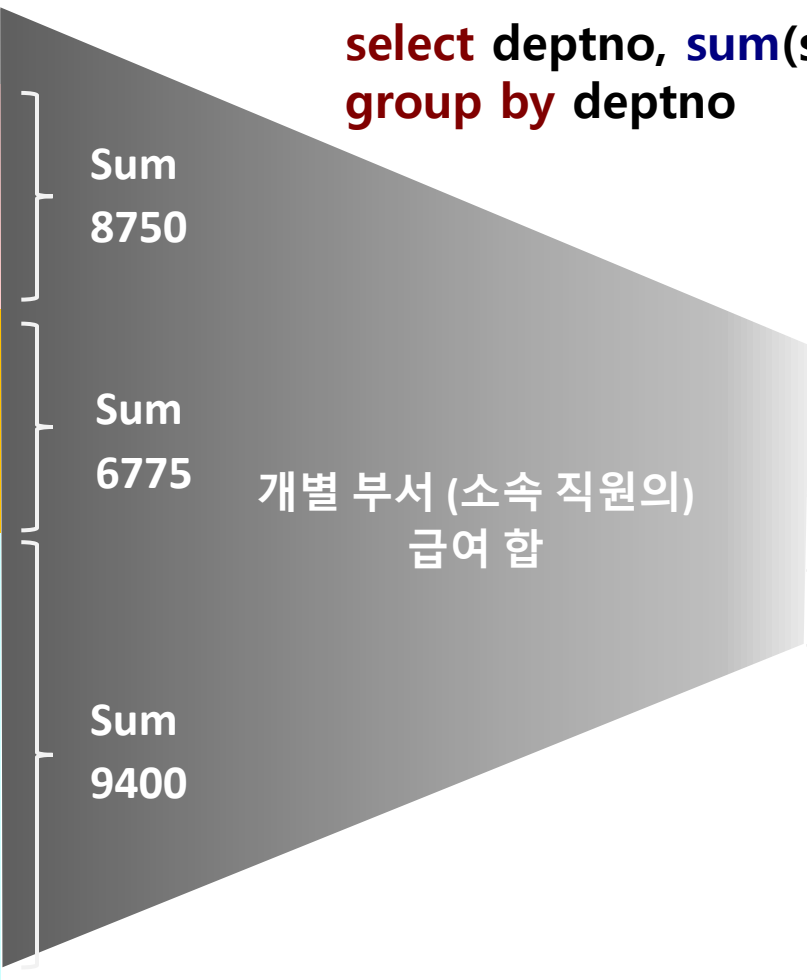
```
select to_timestamp('2022-01-01 14:36:52', 'yyyy-mm-dd hh24:mi:ss') + interval '7 hour'
```

↓
2022-01-01 21:36:52.000

Group by와 집계 함수(Aggregate Function)

Group by 개요

empno	ename	deptno	sal
7934	MILLER	10	1300
7782	CLARK	10	2450
7839	KING	10	5000
7566	JONES	20	2975
7902	FORD	20	3000
7369	SMITH	20	800
7499	ALLEN	30	1600
7521	WARD	30	1250
7844	TURNER	30	1500
7900	JAMES	30	950
7654	MARTIN	30	1250
7698	BLAKE	30	2850



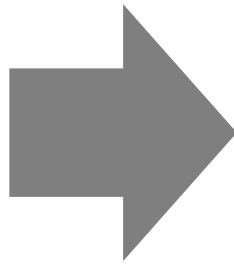
select deptno, **sum**(sal) **as** sum_salary **from** hr.emp
group by deptno

deptno	Sum_salary
10	8750
20	6775
30	9400

Group by 이해

- Group by 절에 기술된 컬럼 값(또는 가공 컬럼값)으로 그룹화 한 뒤 집계(Aggregation) 함수와 함께 사용되어 그룹화된 집계 정보를 제공
- Group by 절에 기술된 컬럼 값으로 **반드시 1의 집합을** 가지게 됨.
- Select 절에는 Group by 절에 기술된 컬럼(또는 가공 컬럼)과 집계 함수만 사용될 수 있음.

empno	ename	deptno	sal
7934	MILLER	10	1300
7782	CLARK	10	2450
7839	KING	10	5000
7566	JONES	20	2975
7902	FORD	20	3000
7369	SMITH	20	800
7499	ALLEN	30	1600
7521	WARD	30	1250
7844	TURNER	30	1500
7900	JAMES	30	950
7654	MARTIN	30	1250
7698	BLAKE	30	2850



```
select deptno, count(*) as emp_count
      , sum(sal) as sum_sal, avg(sal) as avg_sal, max(sal) as max_sal
from hr.emp
group by deptno
```

deptno	emp_count	sum_sal	avg_sal	max_sal
10	3	8750	2916.667	5000
20	3	6775	2258.333	3000
30	6	9400	1566.667	2850

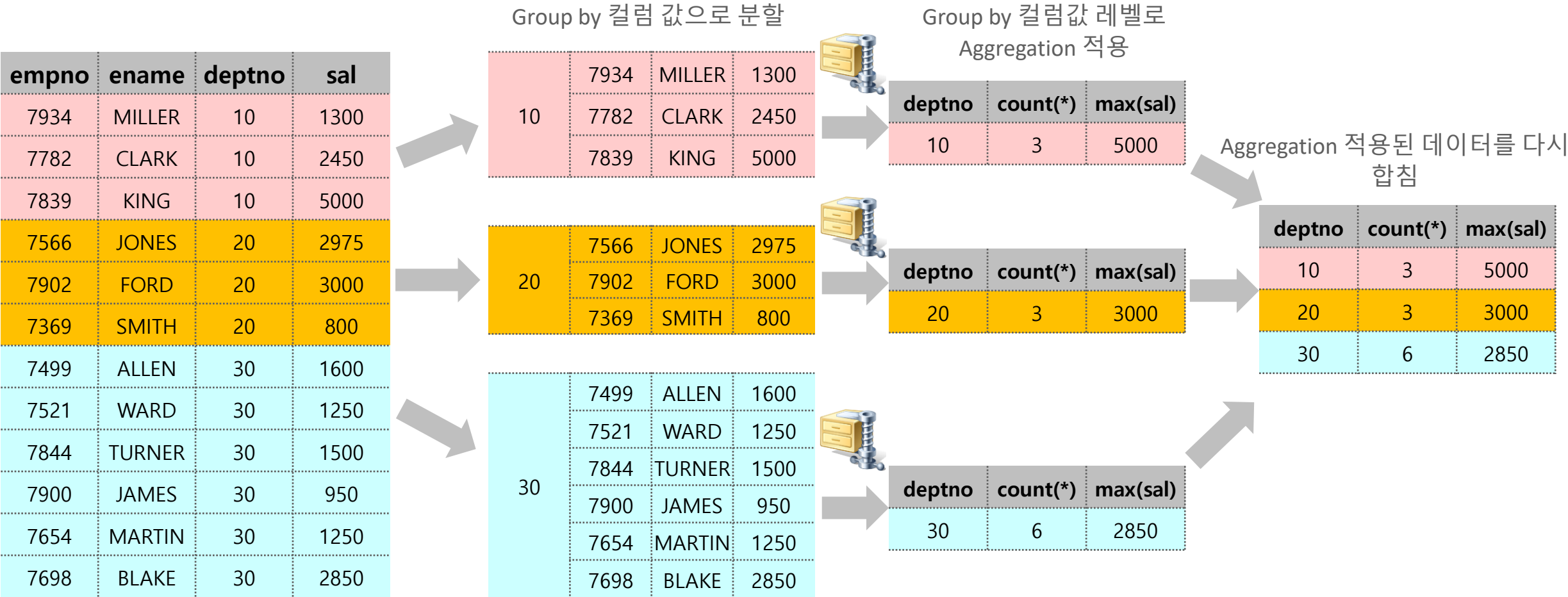
Group by 구문

- SELECT <column(s)>
FROM <table>
WHERE <condition>
GROUP BY <column(s)> -- group by 적용할 컬럼명
HAVING <condition> -- group by 결과의 filtering 조건
ORDER BY <column(s)>;

```
select deptno, sum(sal) as sum_salary  
from hr.emp  
where job != 'SALARYMAN'  
group by deptno  
order by deptno
```

Group by 적용 로직

```
select deptno, count(*) as emp_count, max(sal) as max_sal from hr.emp group by deptno
```



집계 함수(Aggregation Function)의 이해

집계 함수 유형	설명
Count(*)	정해진 집합 레벨에서 데이터 건수를 가져옴
Count(distinct 컬럼명)	정해진 집합 레벨에서 해당 컬럼으로 <u>중복을 배제하고 고유한 건수를 가져옴</u>
Sum(컬럼명)	정해진 집합 레벨에서 지정된 컬럼값의 총합을 가져옴
Min(컬럼명)	정해진 집합 레벨에서 지정된 컬럼값의 최소값을 가져옴
Max(컬럼명)	정해진 집합 레벨에서 지정된 컬럼값의 최대값을 가져옴
Avg(컬럼명)	정해진 집합 레벨에서 지정된 컬럼값의 평균값을 가져옴

집계 함수 적용 시 유의 사항

- 집계 함수는 **Null**을 계산하지 않음

- Min, Max 함수의 경우 숫자값 뿐만 아니라 문자열, 날짜/시간형도 가능

Min, Max 가능
예) $\rightarrow \text{MAX}['ABC', 'BCD']$

$\text{AVG}[7, \text{NULL}, 5, 3] = 5: \text{null 제외}$

$$\begin{aligned}\text{AVG}[4, 0, 5, 2, \text{NULL}] &= (4+0+5+2)/4 \\ &= \frac{11}{4} = 2.75\end{aligned}$$

$\text{max}(\text{date}) = \text{가장 최근}$
 $\text{min}(\text{date}) = \text{가장 나중}$

Group by count(distinct) 케이스

sum or AVG는 문맥이
적용X

orders

order_id	product_id	user_id	order_number
2539329	14084	1	1
2539329	26405	1	1
2539329	12427	1	1
2539329	196	1	1
2539329	26088	1	1
2398795	196	1	2
2398795	12427	1	2
2398795	13176	1	2
2398795	10258	1	2
2398795	26088	1	2
2398795	13032	1	2

중복

select user_id, count(*) as cnt from orders group by user_id

user_id	cnt
1	11

이제 없으면 // 나옴

select user_id, count(distinct product_id) as cnt from orders group by user_id

user_id	cnt
1	8

select count(distinct order_id) as cnt from orders group by user_id

user_id	cnt
1	2

Group by와 Aggregate 함수의 case when을 이용한 피벗팅(Pivoting)

Group by 시 행 레벨로 만들어진 데이터를 열 레벨로 전환할 때 Aggregate와 case when을 결합하여 사용

Year	Month	Revenue
2016	1	10
2016	2	25
2016	3	40
2016
2016	12	78

```
Select Year, Month
      , sum(revenue) as Revenue
From sales
Group by Year, Month
```



Year	Year_rev	Jan	Feb	Mar	...	Dec
2016	450	10	25	40	...	78

```
Select Year, sum(revenue) as Year_rev
      , sum(case when Month = 1 then revenue end) as Jan
      , sum(case when Month = 2 then revenue end) as Feb
      , sum(case when Month = 3 then revenue end) as Mar
      .....
      , sum(case when Month = 12 then revenue end) as Dec
From sales
Group by Year
```

Group by Rollup과 Cube

- Rollup과 Cube는 Group by와 함께 사용되어 Group by 절에 사용되는 컬럼들에 대해서 추가적인 Group by를 수행
- Rollup은 계층적인 방식으로 Group by 추가 수행
- Cube는 Group by 절에 기재된 컬럼들의 가능한 combination으로 Group by 수행.

```
select deptno, job, sum(sal)
from hr.emp
group by rollup(deptno, job)
order by 1, 2;
```

```
select deptno, job, sum(sal)
from hr.emp
group by cube(deptno, job)
order by 1, 2;
```

Group by Rollup의 이해

Group by 시 Rollup을 함께 사용하면 Rollup에 적용된 컬럼의 순서대로 계층적인 Group by 를 추가적으로 수행.

```
select deptno, job, sum(sal)
from hr.emp
group by rollup(deptno, job)
order by 1, 2;
```

(dept, job)
↓
(dept)
↓
()

deptno	job	sum
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
10	Null	8750
20	ANALYST	3000
20	CLERK	800
20	MANAGER	2975
20	Null	6775
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600
30	Null	9400
Null	Null	24925

dept + job 레벨 Group by

dept 레벨 Group by

dept + job 레벨 Group by

dept 레벨 Group by

dept + job 레벨 Group by

dept 레벨 Group by

전체 Aggregation

Group by Cube의 이해

Group by 시 Cube를 함께 사용하면
Cube에 나열된 컬럼들의 가능한 결합으로
Group by 수행.

```
select deptno, job, sum(sal)
from hr.emp
group by cube(deptno, job)
order by 1, 2;

(dept, job)

(dept)

(job)

( )
```

deptno	job	sum	
10	CLERK	1300	dept + job 레벨 Group by
10	MANAGER	2450	
10	PRESIDENT	5000	
10	Null	8750	dept 레벨 Group by
20	ANALYST	3000	dept + job 레벨 Group by
20	CLERK	800	
20	MANAGER	2975	
20	Null	6775	dept 레벨 Group by
30	CLERK	950	dept + job 레벨 Group by
30	MANAGER	2850	
30	SALESMAN	5600	
30	Null	9400	dept 레벨 Group by
Null	ANALYST	3000	Job 레벨 Group by
Null	CLERK	3050	
Null	MANAGER	8275	
Null	PRESIDENT	5000	
Null	SALESMAN	5600	
Null	Null	24925	전체 Aggregation

Group by Rollup과 Cube의 적용

Rollup(YEAR, MONTH, DAY)

YEAR, MONTH, DAY

YEAR, MONTH

YEAR

()

Group by 절의 나열된 컬럼수가 N개 이면

Group by는 N+1회 수행

Cube(YEAR, MONTH, DAY)

YEAR, MONTH, DAY

YEAR, MONTH

YEAR, DAY

YEAR

MONTH, DAY

MONTH

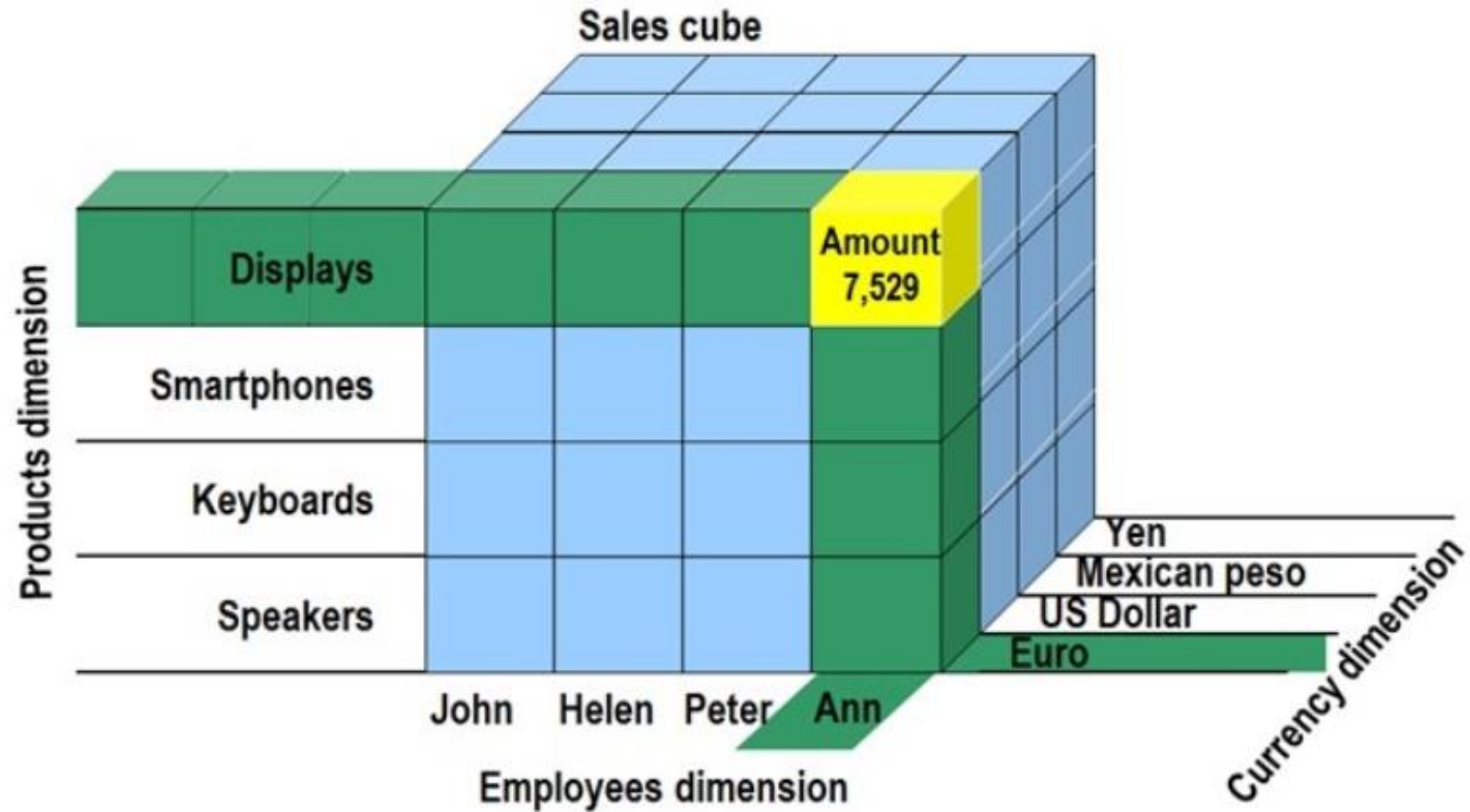
DAY

()

Group by 절의 나열된 컬럼수가 N개 이면

Group by는 2^N 회 수행

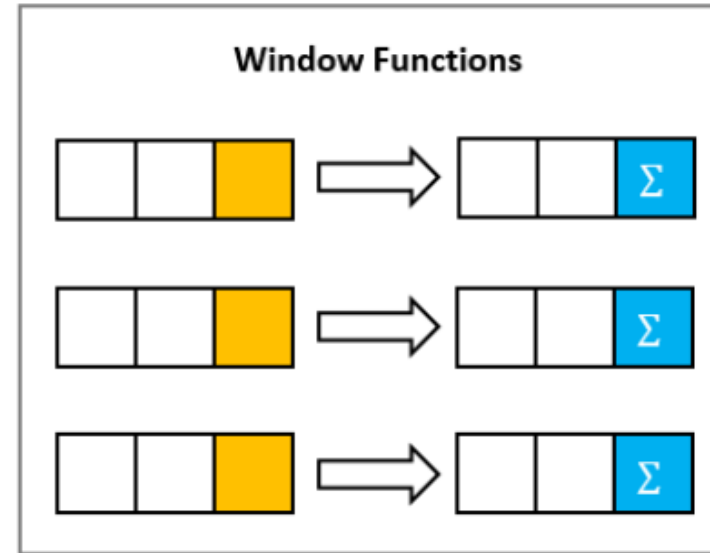
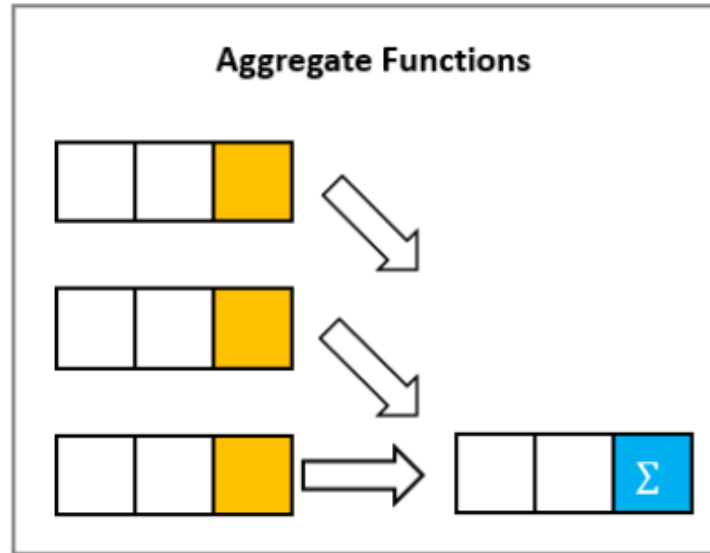
Cube의 이해



Analytic SQL



Analytic SQL – Group by 집계 함수 차이



- Group by 는 원본 데이터 집합의 레벨을 변경하여 적용
- Analytic SQL은 원본 데이터 집합의 레벨을 그대로 유지하면서 적용
- Window를 이용하여 Row 단위의 집합 연산 수행 가능

Analytic SQL 유형 및 사용법

유형	함수
순위/비율 함수	Rank, dense_rank, row_number Percent_rank, cume_dist, ntile
집계(Aggregate) 함수	sum, max, min, avg, count
Lead/Lag	Lead, Lag
First_value/Last_value	First_value, Last_value
Inverse Percentile	Percentile_cont, Percentile_disc

<Analytic function> (인자1, ...)

OVER (

[Partition 절]

[Sorting 절]

[window 절]

)

Analytic SQL 적용 로직



<https://www.kaggle.com/alexisbcook/analytic-functions>

Analytic SQL 예시

```
select a.*,  
sum(sal) over(partition by deptno order by hiredate rows between unbounded preceding and current row) as sum  
from hr.emp a;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	sum
7782	CLARK	MANAGER	7839	1981-06-09	2450		10	2450
7839	KING	PRESIDENT		1981-11-17	5000		10	7450
7934	MILLER	CLERK	7782	1982-01-23	1300		10	8750
7369	SMITH	CLERK	7902	1980-12-17	800		20	800
7566	JONES	MANAGER	7839	1981-04-02	2975		20	3775
7902	FORD	ANALYST	7566	1981-12-03	3000		20	6775
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	1600
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	2850
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30	5700
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	7200
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30	8450
7900	JAMES	CLERK	7698	1981-12-03	950		30	9400

Analytic SQL 특징

<Analytic function> (인자1, ...)

OVER (

[Partition 절]



그룹화 컬럼명

[Sorting 절]



정렬 컬럼명(Window 이동 방향 기준 컬럼명)

[window 절]



Window 범위(Row, Range)

)

!!! 절의 순서가 바뀌어서는 안됨.

원본 데이터의 레벨을 그대로 유지하면서,
그룹핑 레벨에서 자유롭게 **Window의 이동**
과 크기를 조절하면서 Analytic 을 수행.

자유로운 window 설정에 따른 analytic 구사가 가능하므로
SQL의 Analytic 함수를 window 함수로도 지칭

순위 Analytic SQL

- 일반적인 순위: rank, dense_rank, row_number
- 0 ~ 1 사이 정규화 순위: cume_dist, percent_rank
- 분위: ntile

순위 Analytic – rank, dense_rank, row_number

- 전체 데이터/특정 그룹핑 내에서 특정 기준으로 순위를 매기는 함수
- Rank, dense_rank, row_number는 **공동 순위**를 정하는 로직이 조금씩 다름.

순위 함수 유형	설명
rank	공동 순위가 있을 경우 다음 순위는 공동 순위 개수만큼 밀려서 정함. 1, 2, 2, 4 또는 1, 2, 2, 2, 5
dense_rank	공동 순위가 있더라도 다음 순위는 바로 이어서 정함 1, 2, 2, 3 또는 1, 2, 2, 2, 3
row_number	공동 순위가 있더라도 반드시 unique한 순위를 정함 1, 2, 3, 4, 5

rank, dense_rank, row_number 사용하기 - 1

```
select a.empno, ename, job, sal
      , rank() over(order by sal desc) as rank
      , dense_rank() over(order by sal desc) as dense_rank
      , row_number() over (order by sal desc) as row_number from hr.emp a;
```

empno	ename	job	sal	rank	dense_rank	row_number
7839	KING	PRESIDENT	5000	1	1	1
7902	FORD	ANALYST	3000	2	2	2
7566	JONES	MANAGER	2975	3	3	3
7698	BLAKE	MANAGER	2850	4	4	4
7782	CLARK	MANAGER	2450	5	5	5
7499	ALLEN	SALESMAN	1600	6	6	6
7844	TURNER	SALESMAN	1500	7	7	7
7934	MILLER	CLERK	1300	8	8	8
7654	MARTIN	SALESMAN	1250	9	9	9
7521	WARD	SALESMAN	1250	9	9	10
7900	JAMES	CLERK	950	11	10	11
7369	SMITH	CLERK	800	12	11	12

rank, dense_rank, row_number 사용하기 - 2

```
select a.empno, ename, job, deptno, sal
, rank() over(partition by deptno order by sal desc) as rank
, dense_rank() over(partition by deptno order by sal desc) as dense_rank
, row_number() over (partition by deptno order by sal desc) as row_number from hr.emp a;
```

empno	ename	job	deptno	sal	rank	dense_rank	row_number
7839	KING	PRESIDENT	10	5000	1	1	1
7782	CLARK	MANAGER	10	2450	2	2	2
7934	MILLER	CLERK	10	1300	3	3	3
7902	FORD	ANALYST	20	3000	1	1	1
7566	JONES	MANAGER	20	2975	2	2	2
7369	SMITH	CLERK	20	800	3	3	3
7698	BLAKE	MANAGER	30	2850	1	1	1
7499	ALLEN	SALESMAN	30	1600	2	2	2
7844	TURNER	SALESMAN	30	1500	3	3	3
7521	WARD	SALESMAN	30	1250	4	4	4
7654	MARTIN	SALESMAN	30	1250	4	4	5
7900	JAMES	CLERK	30	950	6	5	6

순위 함수 실습 퀴즈

- 회사내 근무 기간 순위(hiredate) : 공동 순위가 있을 경우 차순위는 밀려서 순위 정함
- 부서별로 가장 급여가 높은/낮은 순으로 순위: 공동 순위 시 차순위는 밀리지 않음.
- 부서별 가장 급여가 높은 직원 정보: 공동 순위는 없으며 반드시 고유 순위를 정함.
- 부서별 급여 top 2 직원 정보: 공동 순위는 없으며 반드시 고유 순위를 정함.
- 부서별 가장 급여가 높은 직원과 가장 급여가 낮은 직원 정보. 공동 순위는 없으며 반드시 고유 순위를 정함.
- 회사내 커미션 순위

순위 함수에서 NULL 처리 로직

rank () OVER (

< Partition 절>

order by column [nulls first/last]

)

Nulls first는 Null을 최우선 순위로 ,
Nulls last는 Null을 가장 마지막 순위로 설정.

순위 함수에서 NULL 처리 로직 – nulls first

```
select a.*
```

```
, rank() over (order by comm desc nulls first) as comm_rank
```

```
from hr.emp a;
```

Order by 시 기본적으로 생략되어 있으며 Default는 nulls first임.

Nulls first는 Null을 최우선으로, **Nulls last**는 Null을 가장 마지막으로

empno	ename	job	mgr	hiredate	sal	comm	deptno	comm_rank
7934	MILLER	CLERK	7782	1982-01-23	1300		10	1
7839	KING	PRESIDENT		1981-11-17	5000		10	1
7900	JAMES	CLERK	7698	1981-12-03	950		30	1
7902	FORD	ANALYST	7566	1981-12-03	3000		20	1
7369	SMITH	CLERK	7902	1980-12-17	800		20	1
7566	JONES	MANAGER	7839	1981-04-02	2975		20	1
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30	1
7782	CLARK	MANAGER	7839	1981-06-09	2450		10	1
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30	9
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	10
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	11
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	12

순위 함수에서 NULL 처리 로직 – nulls last

select a.*

, rank() over (order by comm desc **nulls last**) as comm_rank

from hr.emp a;

empno	ename	job	mgr	hiredate	sal	comm	deptno	comm_rank
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30	1
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	2
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	3
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	4
7839	KING	PRESIDENT		1981-11-17	5000		10	5
7900	JAMES	CLERK	7698	1981-12-03	950		30	5
7902	FORD	ANALYST	7566	1981-12-03	3000		20	5
7369	SMITH	CLERK	7902	1980-12-17	800		20	5
7934	MILLER	CLERK	7782	1982-01-23	1300		10	5
7566	JONES	MANAGER	7839	1981-04-02	2975		20	5
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30	5
7782	CLARK	MANAGER	7839	1981-06-09	2450		10	5

집계(Aggregate) Analytic SQL

sum(), max(), min(), avg(), count() 와 같은 집계 함수를 window를 이용하여 로우 레벨로 자유 자재로 집계할 수 있는 기능 제공.

<Analytic function> (인자1, ...)

OVER (

[Partition 절]



그룹화 컬럼명

[Sorting 절]



정렬 컬럼명(Window 이
동 방향 기준 컬럼명)

[window 절]



Window 범위(Rows, Range)

)

sum() Analytic SQL 활용- 1

order_id 별 amount 총합

```
select order_id, line_prod_seq, product_id, amount  
, sum(amount) over (partition by order_id) as total_sum_by_ord from nw.order_items
```

					Partition by order_id
order_id	line_prod_seq	product_id	amount	total_sum_by_ord	
10248	1	11	168	440	
10248	2	42	98	440	
10248	3	72	174	440	
10249	1	14	167.4	1863.4	
10249	2	51	1696	1863.4	
10250	1	41	77	1552.6	
10250	2	51	1261.4	1552.6	
10250	3	65	214.2	1552.6	

sum() Analytic SQL 활용- 2

order_id별 line_prod_seq순으로 누적 amount 합

```
select order_id, line_prod_seq, product_id, amount
      , sum(amount) over (partition by order_id) as total_sum_by_ord
      , sum(amount) over (partition by order_id order by line_prod_seq) as cum_sum_by_ord
from nw.order_items;
```

order_id	line_prod_seq	product_id	amount	total_sum_by_ord	cum_sum_by_ord
10248	1	11	168	440	168
10248	2	42	98	440	266
10248	3	72	174	440	440
10249	1	14	167.4	1863.4	167.4
10249	2	51	1696	1863.4	1863.4
10250	1	41	77	1552.6	77
10250	2	51	1261.4	1552.6	1338.4
10250	3	65	214.2	1552.6	1552.6

sum() Analytic SQL 활용- 3

```
select order_id, line_prod_seq, product_id, amount  
, sum(amount) over (partition by order_id) as total_sum_by_ord
```

```
, sum(amount) over (partition by order_id order by line_prod_seq) as cum_sum_by_ord_01
```

```
, sum(amount) over (partition by order_id order by line_prod_seq rows between unbounded preceding and current row) as cum_sum_by_ord_02
```

```
, sum(amount) over ( ) as total_sum
```

```
from nw.order_items where order_id between 10248 and 10250;
```

partition 또는 order by 절이 없을 경우 window

order_id	line_prod_seq	product_id	amount	total_sum_by_ord	cum_sum_by_ord_01	cum_sum_by_ord_02	total_sum
10248	1	11	168	440	168	168	3856
10248	2	42	98	440	266	266	3856
10248	3	72	174	440	440	440	3856
10249	1	14	167.4	1863.4	167.4	167.4	3856
10249	2	51	1696	1863.4	1863.4	1863.4	3856

- 집계(aggregate) 계열 analytic 함수는 order by 절이 있을 경우 window 절은 기본적으로 range unbounded preceding and current row 임
- 만약 order by 절이 없다면 window는 해당 partition의 모든 row를 대상
- 만약 partition 절도 없다면 window는 전체 데이터의 row를 대상.

max() Analytic SQL 활용

order_id 별 상품 최대 구매금액, order_id별 상품 누적 최대 금액

```
select order_id, line_prod_seq, product_id, amount
      , max(amount) over (partition by order_id) as max_by_ord
      , max(amount) over (partition by order_id order by line_prod_seq) as cum_max_by_ord
from nw.order_items;
```

order_id	line_prod_seq	product_id	amount	max_by_ord	cum_max_by_ord
10248	1	11	168	174	168
10248	2	42	98	174	168
10248	3	72	174	174	174
10249	1	14	167.4	1696	167.4
10249	2	51	1696	1696	1696
10250	1	41	77	1261.4	77
10250	2	51	1261.4	1261.4	1261.4
10250	3	65	214.2	1261.4	1261.4

min() Analytic SQL 활용

order_id 별 상품 최소 구매금액, order_id별 상품 누적 최소 금액

```
select order_id, line_prod_seq, product_id, amount
      , min(amount) over (partition by order_id) as max_by_ord
      , min(amount) over (partition by order_id order by line_prod_seq) as cum_max_by_ord
from nw.order_items;
```

order_id	line_prod_seq	product_id	amount	min_by_ord	cum_min_by_ord
10248	1	11	168	98	168
10248	2	42	98	98	98
10248	3	72	174	98	98
10249	1	14	167.4	167.4	167.4
10249	2	51	1696	167.4	167.4
10250	1	41	77	77	77
10250	2	51	1261.4	77	77
10250	3	65	214.2	77	77

avg() Analytic SQL 활용

order_id 별 상품 평균 구매금액, order_id별 상품 누적 평균 구매금액

```
select order_id, line_prod_seq, product_id, amount
      , avg(amount) over (partition by order_id) as avg_by_ord
      , avg(amount) over (partition by order_id order by line_prod_seq) as cum_avg_by_ord
from nw.order_items;
```

order_id	line_prod_seq	product_id	amount	avg_by_ord	cum_avg_by_ord
10248	1	11	168	146.6667	168
10248	2	42	98	146.6667	133
10248	3	72	174	146.6667	146.6667
10249	1	14	167.4	931.7	167.4
10249	2	51	1696	931.7	931.7
10250	1	41	77	517.5333	77
10250	2	51	1261.4	517.5333	669.2
10250	3	65	214.2	517.5333	517.5333

count() Analytic SQL 활용

order_id 별 건수, order_id별 누적(?) 건수

```
select order_id, line_prod_seq, product_id, amount
      , count(line_prod_seq) over (partition by order_id) as cnt_by_ord
      , count(line_prod_seq) over (partition by order_id order by line_prod_seq) as cum_cnt_by_ord
from nw.order_items;
```

order_id	line_prod_seq	product_id	amount	cnt_by_ord	cum_cnt_by_ord
10248	1	11	168	3	1
10248	2	42	98	3	2
10248	3	72	174	3	3
10249	1	14	167.4	2	1
10249	2	51	1696	2	2
10250	1	41	77	3	1
10250	2	51	1261.4	3	2
10250	3	65	214.2	3	3

Aggregate analytic SQL 실습

- 직원 정보 및 부서별로 직원 급여의 hiredate 순으로 누적 급여 합
- 직원 정보 및 부서별 평균 급여와 개인 급여와의 차이 출력
- 직원 정보 및 부서별 총 급여 대비 개인 급여의 비율 출력(소수점 2자리 까지로 비율 출력)
- 직원 정보 및 부서에서 가장 높은 급여 대비 비율 출력(소수점 2자리 까지로 비율 출력)
- product_id 총 매출액을 구하고, 전체 매출 대비 개별 상품의 총 매출액 비율을 소수점2자리로 구한 뒤 매출액 비율 내림차순으로 정렬
- 직원별 개별 상품 매출액, 직원별 전체 상품 매출액을 구하고, 직원별로 가장 높은 매출을 올리는 상품의 매출 금액 대비 개별 상품 매출 비율 구하기
- 상품별 매출합을 구하되, 상품 카테고리별 매출합의 5% 이상이고, 동일 카테고리에서 상위 3개의 상품 정보 추출.

Window 절 구문

```
windowing_clause =  
    { ROWS | RANGE }  
    { BETWEEN  
        { UNBOUNDED PRECEDING | CURRENT ROW | value_expr { PRECEDING | FOLLOWING }  
    }  
    AND  
    { UNBOUNDED FOLLOWING | CURRENT ROW | value_expr { PRECEDING | FOLLOWING }  
    }  
    | { UNBOUNDED PRECEDING | CURRENT ROW | value_expr PRECEDING }  
    }
```


Window 절의 상세 구문 설명

구문	구문 설명
ROWS RANGE	<p>Window의 개별 row를 정의함. Rows는 물리적인 row를, Range는 논리적인 row를 의미. Order by 절이 없으면 해당 구문은 기술할 수 없음.</p> <ul style="list-style-type: none">over (partition by category_id order by unit_price rows between unbounded preceding and current row)over (partition by customer_id order by order_date range between interval '2' day preceding and current row)
BETWEEN ... AND	<p>Window의 시작과 종료 지점을 기술. Between 다음이 시작 지점, And 다음이 종료 지점.</p> <p>Between 이 없다면 Row Range 다음이 시작점, (기본 설정으로) 현재 Row(Current row)가 종료점으로 설정.</p> <p>over (partition by category_id order by unit_price rows between 2 preceding and current row)</p> <p>over (partition by category_id order by unit_price rows 2 preceding)</p>
UNBOUNDED PRECEDING	<p>Window의 시작이 Partition의 첫번째 row부터 시작함을 기술. Window의 종료점으로는 사용될 수 없음.</p>
UNBOUNDED FOLLOWING	<p>Window의 종료가 Partition의 마지막 row에서 종료됨을 기술. Window의 시작점으로는 사용될 수 없음.</p>
CURRENT ROW	<p>Window의 시작점 또는 종료점으로 사용될 수 있으나, 보통은 종료점으로 사용.</p> <p>종료점으로 사용시 window의 종료가 현재 row에서 종료됨을 기술.</p> <p>시작점으로 사용시 window의 시작이 현재 row에서 시작됨을 기술</p> <ul style="list-style-type: none">over (partition by category_id order by unit_price rows between unbounded preceding and current row)over (partition by category_id order by unit_price rows between current row and unbounded following)

Window 절의 상세 구문 설명

구문	구문 설명
ROWS + value 표현 PRECEDING or value 표현 FOLLOWING	<p>2 preceding 또는 3 following과 같이 value 표현 preceding/following을 지정하여 특정 지점을 window의 시작 또는 종료 지점으로 나타낼 수 있음. Rows나 Range 모두와 함께 사용될 수 있으며, Rows와 사용될 때는 물리적인 row위치를 지정</p> <ul style="list-style-type: none">• over (partition by category_id order by unit_price rows between 2 preceding and current row)• over (partition by category_id order by unit_price rows between 2 preceding and 1 following)• over (partition by category_id order by unit_price rows between and 1 following and current row)
RANGE + value 표현 PRECEDING or value 표현 FOLLOWING	<p>마찬가지로 특정 지점을 window의 시작 또는 종료 지점으로 나타낼 수 있음. Range와 사용할 때는 논리적인 row 위치를 지정하므로 value 표현이 단순한 숫자외에도 논리적인 값을 적용되어야 함. 보통은 숫자값과 interval 값이 사용됨. 또한 order by 절의 컬럼도 numeric 또는 (대부분) date/timestamp 가 되어야 함.</p> <ul style="list-style-type: none">• over (partition by customer_id order by order_date range between interval '2' day preceding and current row)

Window 의 범위 설정 - rows between unbounded preceding and current row

`select *, sum(unit_price) over (order by unit_price) as unit_price_sum from products;`

`select *, sum(unit_price) over (order by unit_price rows between unbounded preceding and current row) from products;`

`select *, sum(unit_price) over (order by unit_price rows unbounded preceding) as unit_price_sum from products;`

product_id	product_name	category_id	unit_price	unit_price_sum
33	Geitost	4	2.5	2.5
24	Guaran Fantstica	1	4.5	7
13	Konbu	8	6	13
52	Filo Mix	5	7	20
54	Tourtire	6	7.45	27.45
75	Rhnbru Klosterbier	1	7.75	35.2

- 집계(aggregate) 계열 analytic 함수는 order by 절이 있을 경우 window 절은 기본적으로 range between unbounded preceding and current row
- Windows 절에 between이 주어지지 않고 시작 범위만 주어질 때 종료 범위는 자동으로 current row .

Window 의 범위 설정 - 중앙합 또는 중앙 평균(Centered average)

중앙합 또는 중앙 평균(Centered average)

```
select product_id, product_name, category_id, unit_price
      , sum(unit_price) over (partition by category_id order by unit_price rows between 1 preceding and 1 following) as unit_price_sum
from products
```

product_id	product_name	category_id	unit_price	unit_price_sum
24	Guaran Fantstica	1	4.5	12.25
75	Rhnbru Klosterbier	1	7.75	26.25
34	Sasquatch Ale	1	14	35.75
67	Laughing Lumberjack Lager	1	14	43
70	Outback Lager	1	15	47
35	Steeleye Stout	1	18	51
76	Lakkalikri	1	18	54
1	Chai	1	18	54

Window 의 범위 설정 - rows between unbounded preceding and unbounded following

select product_id, product_name, category_id, unit_price
, sum(unit_price) over (partition by category_id order by unit_price **rows between unbounded preceding and unbounded following**) as
unit_price_sum from products

product_id	product_name	category_id	unit_price	unit_price_sum
24	Guaran Fantstica	1	4.5	455.75
75	Rhnbru Klosterbier	1	7.75	455.75
34	Sasquatch Ale	1	14	455.75
67	Laughing Lumberjack Lager	1	14	455.75
70	Outback Lager	1	15	455.75
35	Steeleye Stout	1	18	455.75
76	Lakkalikri	1	18	455.75
1	Chai	1	18	455.75
39	Chartreuse verte	1	18	455.75
2	Chang	1	19	455.75
43	Ipoh Coffee	1	46	455.75
38	Cte de Blaye	1	263.5	455.75

Window 의 범위 설정 - rows between current row and unbounded following

select product_id, product_name, category_id, unit_price
, sum(unit_price) over (partition by category_id order by unit_price **rows between current row and unbounded following**) as unit_price_sum
from products;

product_id	product_name	category_id	unit_price	unit_price_sum
24	Guaran Fantstica	1	4.5	455.75
75	Rhnbru Klosterbier	1	7.75	451.25
34	Sasquatch Ale	1	14	443.5
67	Laughing Lumberjack Lager	1	14	429.5
70	Outback Lager	1	15	415.5
35	Steeleye Stout	1	18	400.5
76	Lakkalikri	1	18	382.5
1	Chai	1	18	364.5
39	Chartreuse verte	1	18	346.5
2	Chang	1	19	328.5
43	Ipoh Coffee	1	46	309.5
38	Cte de Blaye	1	263.5	263.5

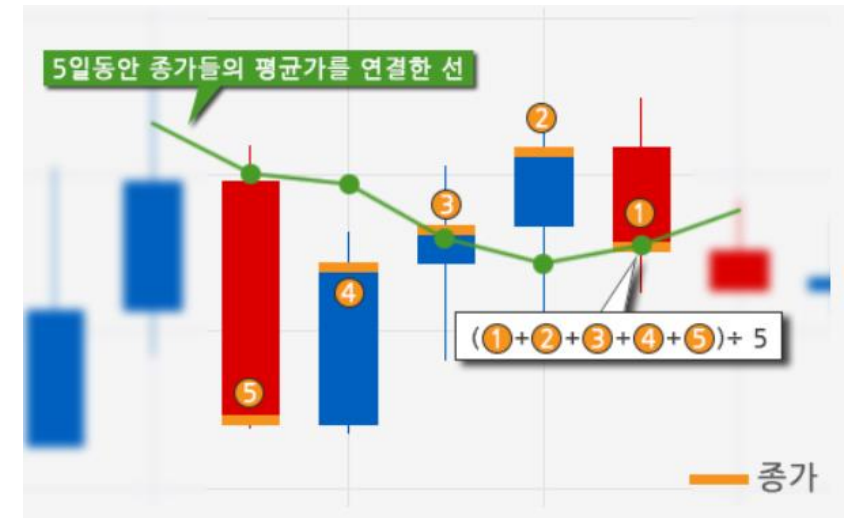
Window 의 범위 설정 - range와 rows의 차이

```
select *  
, sum(sum_by_daily_cat) over (partition by category_id order by ord_date rows between 2 preceding and current row) as sum_rows  
, sum(sum_by_daily_cat) over (partition by category_id order by ord_date range between interval '2' day preceding and current row) as sum_range  
from temp_01;
```

category_id	ord_date	sum_by_daily_cat	sum_rows	sum_range
1	1996-07-10	604.8	604.8	604.8
1	1996-07-11	45.9	650.7	650.7
1	1996-07-12	304	954.7	954.7
1	1996-07-16	86.4	436.3	86.4
1	1996-07-17	608	998.4	694.4
1	1996-07-19	477	1171.4	1085
1	1996-07-23	100.8	1185.8	100.8
1	1996-07-24	532	1109.8	632.8
1	1996-07-25	240	872.8	872.8

이동 평균(Moving Average)

평균을 구하는 범위 구간을 이동 시키면서 구하는 평균값.



이동 평균

```
select ord_date, daily_sum  
      , avg(daily_sum) over (order by ord_date rows between 2 preceding and current row) as ma_3days  
from temp_01
```

ord_date	daily_sum	ma_3days
1996-07-04	440	440
1996-07-05	1863.4	1151.7
1996-07-08	2206.66	1503.353
1996-07-09	3597.9	2555.987
1996-07-10	1444.8	2416.453
1996-07-11	556.62	1866.44
1996-07-12	2490.5	1497.307
1996-07-15	517.8	1188.307
1996-07-16	1119.9	1376.067
1996-07-17	1614.88	1084.193

range와 rows 적용 시 유의 사항

- range는 논리적인 row 위치를 지정하므로 보통은 숫자값과 interval값으로 window의 크기를 설정함.
over (partition by customer_id order by order_date **range between interval '2' day preceding and current row**)
- 또한 range는 rows와 동일한 window 크기 syntax도 사용 가능함(unbounded preceding, unbounded following)
- 집계(aggregate) 계열 analytic 함수는 order by 절이 있을 경우 window 절은 기본적으로 **range between unbounded preceding and current row**임. 즉 sum(sal) over (order by hiredate) 는 sum(sal) over (order by hiredate range between unbounded preceding and current row)
- 하지만 range를 적용할 경우는 order by에서 동일 값이 있을 경우에 current row를 자신의 row가 아닌 동일 값이 있는 전체 row를 동일 그룹으로 간주하여 집계 analytic을 적용하므로 **rows를 명시적으로 사용하는 경우와 값이 달라질 수 있음.**

range와 rows의 차이 – order by 시 동일 row 처리 차이

```
select empno, deptno, sal
, sum(sal) over (partition by deptno order by sal) as sum_default
, sum(sal) over (partition by deptno order by sal range between unbounded preceding and current row) as sum_range
, sum(sal) over (partition by deptno order by sal rows between unbounded preceding and current row) as sum_rows
from hr.emp
```

empno	deptno	sal	sum_default	sum_range	sum_rows
7934	10	1300	1300	1300	1300
7782	10	2450	3750	3750	3750
7839	10	5000	8750	8750	8750
7369	20	800	800	800	800
7566	20	2975	3775	3775	3775
7902	20	3000	6775	6775	6775
7900	30	950	950	950	950
7521	30	1250	3450	3450	2200
7654	30	1250	3450	3450	3450
7844	30	동일한 order by 값		4950	4950
7499	30	1800	6550	6550	6550
7698	30	2850	9400	9400	9400

range와 rows의 차이 – order by 시 동일 row 처리 차이

```
select empno, deptno, sal, date_trunc('month', hiredate)::date as hiremonth
, sum(sal) over (partition by deptno order by date_trunc('month', hiredate)) as sum_default
, sum(sal) over (partition by deptno order by date_trunc('month', hiredate) rows between unbounded preceding and current row) as sum_rows
from hr.emp;
```

empno	deptno	sal	hiremonth	sum_default	sum_rows
7782	10	2450	1981-06-01	2450	2450
7839	10	5000	1981-11-01	7450	7450
7934	10	1300	1982-01-01	8750	8750
7369	20	800	1980-12-01	800	800
7566	20	2975	1981-04-01	3775	3775
7902	20	3000	1981-12-01	6775	6775
7521	30	1250	1981-02-01	2850	1250
7499	30	1600	1981-02-01	2850	2850
7698	30	2850	1981-05-01	5700	5700
7844	30	1500	1981-09-01	8450	7200
7654	30	1250	1981-09-01	8450	8450
7900	30	950	1981-12-01	9400	9400

LAG() 와 LEAD() Analytic SQL

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

- LAG는 현재 행보다 이전 행의 값을 가져옴
- LEAD는 현재 행보다 다음 행의 값을 가져옴
- LAG()/LEAD() 함수내의 인자로 아래 3개 인자를 입력 받음.
(expr은 반드시 입력, offset과 default는 생략 가능)
 - ✓ expr : 적용할 컬럼명
 - ✓ offset: 값을 가져올 행의 위치 offse값. 기본값은 1임
 - ✓ default: 행의 위치 offse으로 Null값일 때 대체할 값. 기본값은 Null임.
- partition by는 생략 가능하지만, order by 는 반드시 필요함.
- window 절은 사용되지 않습니다.

lag(ename) over (partition by deptno order by hiredate)

empno	deptno	hiredate	ename	prev_ename
7782	10	1981-06-09	CLARK	
7839	10	1981-11-17	KING	CLARK
7934	10	1982-01-23	MILLER	KING
7369	20	1980-12-17	SMITH	
7566	20	1981-04-02	JONES	SMITH
7902	20	1981-12-03	FORD	JONES

lead(ename) over (partition by deptno order by hiredate)

empno	deptno	hiredate	ename	prev_ename
7782	10	1981-06-09	CLARK	KING
7839	10	1981-11-17	KING	MILLER
7934	10	1982-01-23	MILLER	
7369	20	1980-12-17	SMITH	JONES
7566	20	1981-04-02	JONES	FORD
7902	20	1981-12-03	FORD	

LAG() – 현재 행보다 이전 행의 데이터를 가져옴.

```
select empno, deptno, hiredate, ename, lag(ename) over (partition by deptno order by hiredate) as prev_ename  
from emp;
```

* 특별한 이유가 없는 이상 order by는 오름차순으로 고정하는 것이 좋음. orderby 내림차순의 lag()는 orderby 오름차순의 lead()와 유사한 결과를 가져오기 때문

					empno	deptno
					hiredate	ename
					prev_ename	
Partition by deptno	{	7782	10	1981-06-09	CLARK	
		7839	10	1981-11-17	KING	CLARK
		7934	10	1982-01-23	MILLER	KING
	{	7369	20	1980-12-17	SMITH	
		7566	20	1981-04-02	JONES	SMITH
		7902	20	1981-12-03	FORD	JONES

LEAD() – 현재 행보다 다음 행의 데이터를 가져옴

```
select empno, deptno, hiredate, ename, lead(ename) over (partition by deptno order by hiredate) as next_ename
from emp;
```

empno	deptno	hiredate	ename	prev_ename
7782	10	1981-06-09	CLARK	KING
7839	10	1981-11-17	KING	MILLER
7934	10	1982-01-23	MILLER	
7369	20	1980-12-17	SMITH	JONES
7566	20	1981-04-02	JONES	FORD
7902	20	1981-12-03	FORD	

Partition by
deptno

LAG()/LEAD() default값 설정 시 유의 사항

lag()나 lead() 시 window에서 데이터를 가져올 수 없는 위치에 있어서 Null을 반환할 경우 Lag(컬럼명, offset 값, default 값) 과 같이 Null 대신 default 값을 설정할 수 있음. Default 값을 설정 할 경우 offset값이 기본 1이더라도 반드시 값을 1로 명시해 줘야 함.

```
select empno, deptno, hiredate, ename, lag(ename, 1, 'No Previous') over (partition by deptno order by hiredate) as prev_ename from emp;
```

Partition by deptno

empno	deptno	hiredate	ename	prev_ename
7782	10	1981-06-09	CLARK	No Previous
7839	10	1981-11-17	KING	CLARK
7934	10	1982-01-23	MILLER	KING
7369	20	1980-12-17	SMITH	No Previous
7566	20	1981-04-02	JONES	SMITH
7902	20	1981-12-03	FORD	JONES

first_value()/last_value() Analytic SQL

window에서 order by 로 기술된 순으로 가장 첫번째/가장 마지막에 위치한 데이터를 가져옴.

```
FIRST_VALUE / LAST_VALUE ( expression ) OVER (  
    [PARTITION BY partition_expression, ... ] ORDER BY sort_expression [ASC | DESC], ...  
    Window 절  
)
```

- first_value는 window의 가장 첫번째에 위치한 데이터를 가져옴.
- last_value는 window의 가장 마지막에 위치한 데이터를 가져옴. window절이 rows between unbounded preceding and unbounded following이 되어야 함.
- partition by는 생략 가능하지만, **order by** 는 반드시 필요함.
- **window 절은 생략 가능합니다. 생략 시 range between unbounded preceding and current row**

First_value() – 가장 처음 위치의 데이터를 가져오기

부서별로 가장 hiredate가 오래된 사람의 sal 가져오기.

```
select empno, ename, deptno, hiredate, sal  
  , first_value(sal) over (partition by deptno order by hiredate) as first_hiredate_sal from emp;
```

		empno	ename	deptno	hiredate	sal	first_hiredate_sal
Partition by deptno	{	7782	CLARK	10	1981-06-09	2450	2450
		7839	KING	10	1981-11-17	5000	2450
		7934	MILLER	10	1982-01-23	1300	2450
	{	7369	SMITH	20	1980-12-17	800	800
		7566	JONES	20	1981-04-02	2975	800
		7902	FORD	20	1981-12-03	3000	800

Last_value () - 가장 마지막 위치의 데이터를 가져오기

부서별로 가장 hiredate가 최근인 사람의 sal 가져오기. last_value() 사용시에 **rows between unbounded preceding and unbounded following** 을 적용하지 않으면 running last_value() 를 가져오므로 잘못된 결과를 반환할 수 있음에 유의

```
select empno, ename, deptno, hiredate, sal
```

```
, last_value(sal) over (partition by deptno order by hiredate rows between unbounded preceding and unbounded following) as last_hiredate_sal from emp;
```

		empno	ename	deptno	hiredate	sal	last_hiredate_sal
Partition by deptno	{	7782	CLARK	10	1981-06-09	2450	1300
		7839	KING	10	1981-11-17	5000	1300
		7934	MILLER	10	1982-01-23	1300	1300
{		7369	SMITH	20	1980-12-17	800	3000
		7566	JONES	20	1981-04-02	2975	3000
		7902	FORD	20	1981-12-03	3000	3000

Last_value () 적용 시 유의 사항

부서별로 가장 hiredate가 최근인 사람의 sal 가져오기. last_value() 사용시에 **rows between unbounded preceding and unbounded following** 을 적용하지 않으면 running last_value() 를 가져오는 잘못된 결과를 반환할 수 있음에 유의

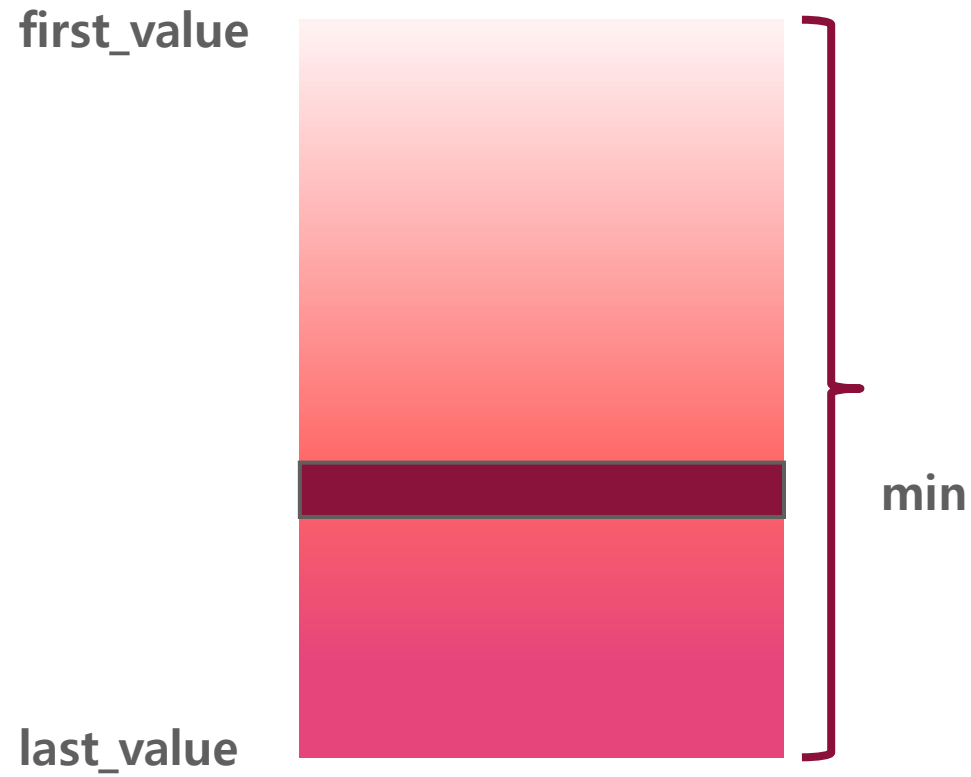
```
select empno, ename, deptno, hiredate, sal
```

```
, last_value(sal) over (partition by deptno order by hiredate rows between unbounded preceding and current row)
```

```
as last_hiredate_sal from emp;
```

Partition by deptno	empno	ename	deptno	hiredate	sal	last_hiredate_sal
	7782	CLARK	10	1981-06-09	2450	2450
	7839	KING	10	1981-11-17	5000	5000
	7934	MILLER	10	1982-01-23	1300	1300
	7369	SMITH	20	1980-12-17	800	800
	7566	JONES	20	1981-04-02	2975	2975
	7902	FORD	20	1981-12-03	3000	3000

first_value와 min Analytic 차이



순위 함수 - cume_dist, percent_rank, ntile

cume_dist()

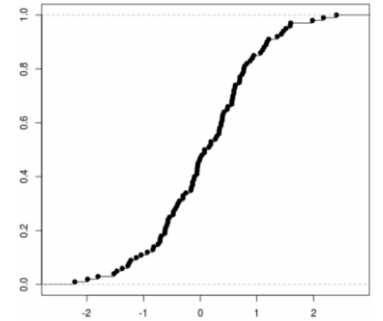
분위수를 파티션내의 건수로 적용하고 0 ~ 1 사이 값으로 변환
파티션 내에서 자신을 포함한 이전 로우수/파티션 내의 로우 건수로 계산

percent_rank()

Rank()를 0 ~ 1 사이 값으로 정규화 시킴
 $(\text{파티션 내의 rank()값} - 1) / (\text{파티션 내의 로우 건수} - 1)$

ntile()

지정된 숫자만큼의 분위를 정하여 그룹핑하는데 적용.



percentile_disc/percentile_cont – Inverse percentile

- percentile_disc/percentile_cont는 inverse percentile이라고도 불리며, 특정 분위수에 해당하는 값을 반환
- percentile_disc는 0 ~ 1 사이의 분위수값을 입력하면 해당 분위수 값 이상인 것 중에서 최소 cume_dist 값을 가지는 값을 반환.
- percentile_disc는 이산 역분포함수, percentile_cont는 연속 역분포함수를 기반으로 함.
- 입력 받은 분위수가 특정 로우를 정확하게 지정하지 못하고, 두 로우 사이일때 percentile_cont는 보간법을 이용하여 보정하며, percentile_disc는 두 로우에서 작은 값을 반환
- postgresql에서는 percentile_cont/percentile_disc는 analytic SQL이 아니라 집계 함수로 사용됨.

서브 쿼리(Subquery)

서브쿼리(Sub-query) 개요

- 서브쿼리는 하나의 쿼리내에 또 다른 쿼리가 포함되어 있는 쿼리를 의미
- 서브쿼리는 메인 쿼리(Main Query)내에 포함되어 있는 관계
- Where절에 사용될 경우 복잡한 업무적인 조건을 직관적인 SQL로 표현하여 필터링하는데 주로 사용됨

```
select * from hr.emp where sal >= (select avg(sal) from hr.emp)
```



서브쿼리

서브쿼리 유형

서브쿼리 유형	예시
Where 절에 사용되는 서브쿼리	<ul style="list-style-type: none">• <code>select * from hr.emp where sal >= (select avg(sal) from hr.emp)</code>• <code>select * from hr.emp_salary_hist a where todate = (select max(todate) from hr.emp_salary_hist b where a.empno = b.empno)</code>
Select 절에 사용되는 서브쿼리 (스칼라 서브쿼리)	<ul style="list-style-type: none">• <code>select ename, deptno, (select dname from hr.dept x where x.deptno=a.deptno) as dname from hr.emp a;</code>
From 절에 사용되는 서브쿼리 (인라인 뷰)	<ul style="list-style-type: none">• <code>select dname from (select * from hr.dept x)</code>

(where절) 서브쿼리의 주요 특징

- 서브쿼리는 메인 쿼리에 where 조건으로 값을 전달하거나 메인쿼리와 연결되어 메인 쿼리의 **필터링** 작업을 수행
 - 서브쿼리 컬럼은 컬럼의 값만 메인쿼리로 전달 될 수 있으며 컬럼 자체는 메인쿼리에서 사용될 수 없음.
 - 메인쿼리의 컬럼은 서브쿼리에서 사용될 수 있음
- 서브쿼리와 메인쿼리로 연결 시 **메인쿼리의 집합 레벨이 변경되지 않음**
 - 메인 쿼리와 서브쿼리와 연결 시 서브쿼리는 연결 컬럼으로 **무조건 유니크한 집합, 즉 1의 집합이 되므로 메인쿼리의 집합 레벨을 변경하지 않음**

(where절) 서브쿼리의 주요 특징 - 1

- 서브쿼리는 메인 쿼리에 where 조건으로만 값을 전달하거나 메인쿼리와 연결되어 메인 쿼리의 **필터링** 작업을 수행
 - 서브쿼리 컬럼은 컬럼의 값만 메인쿼리로 전달 될 수 있으며 컬럼 자체는 메인쿼리에서 사용될 수 없음.
 - 메인쿼리의 컬럼은 서브쿼리에서 사용될 수 있음

```
select a.*  
from hr.dept a where a.deptno in (select deptno from hr.emp x where x.sal > 1000);
```

 OK

```
select a.*, x.ename  
from hr.dept a where a.deptno in (select deptno from hr.emp x where x.sal > 1000 );
```

 수행 안됨


```
select a.* from hr.dept a  
where exists (select deptno from hr.emp x where x.deptno=a.deptno and x.sal > 1000)
```

 OK

(where절) 서브쿼리의 주요 특징 - 2

- 서브쿼리는 유니크한 집합 레벨을 반환.
 - 메인 쿼리와 서브쿼리와 연결 시 서브쿼리는 연결 컬럼으로 **무조건 유니크한 집합, 즉 1의 집합이 됨**
 - 따라서 서브쿼리와 메인쿼리로 연결 시 **메인쿼리의 집합 레벨이 변경되지 않음**

```
select * from nw.orders a where order_id in (select order_id from nw.order_items where amount > 100);
```



1 대 M

Orders와 order_items는 order_id를 기준으로 하면 1:M의 연결관계를 가짐. 하지만 서브쿼리는 메인 쿼리 집합을 변화시키지 않으므로 결과 집합은 orders 집합 레벨임(즉 order_id로 unique집합)

```
select * from nw.orders a where exists (select order_id from nw.order_items x where a.order_id =  
x.order_id and x.amount > 100);
```

서브쿼리와 세미조인(Semi-Join)

- 서브쿼리는 메인쿼리와 연결시에 조인과 유사한 방식(Semi-Join)으로 연결될 수 있음
- 단 서브쿼리는 무조건 1의 집합으로 만들어지므로 메인쿼리 집합 레벨을 변경 시키지 않음

select * from nw.orders a where order_id in (select order_id from nw.order_items where amount > 100)

DB내에서의 실제 실행 계획

Node Type	Entity	Cost	Rows	Time
▼ Hash Join		67.87 - 101.59	782	0.786
Seq Scan	orders	0.00 - 22.30	830	0.069
▼ Hash		57.61 - 57.61	782	0.549
▼ Aggregate		49.40 - 57.61	782	0.496
Seq Scan	order_items	0.00 - 44.94	1783	0.263

Order_items 집합을 **order_id 레벨로 unique하게 1집합으로** 만든 다음에 orders와 조인 수행.

(where절) 서브쿼리의 주요 용도

- 서브쿼리는 메인쿼리의 집합 레벨을 변경 시키지 않으면서 메인 쿼리의 결과 집합을 필터링(또는 체크)하기 위해 많이 사용
- 즉 메인쿼리와 서브쿼리가 연결 조건으로 1:M이 되거나 M:M 이 될 때 메인 쿼리의 집합 레벨을 변경 시키지 않고 서브쿼리로 필터링(또는 체크) 해야할 업무조건이 있을 때 주로 활용

```
select * from nw.orders a where order_id in (select order_id from nw.order_items where amount > 100);
```

```
select * from nw.orders a where exists (select order_id from nw.order_items x where a.order_id = x.order_id and x.amount > 100);
```

서브쿼리 활용 시 유의 사항

- 서브쿼리가 비상관(non-correlated) 서브쿼리인가, 상관(correlated) 서브쿼리인가?
 - ✓ 비상관 서브쿼리: 서브쿼리 자체적으로 메인쿼리에 값을 전달
`select * from hr.dept where deptno in (select deptno from hr.emp where sal < 1300);`
 - ✓ 상관 서브쿼리: 메인쿼리에서 서브쿼리에 연결 컬럼으로 연결한 뒤에 메인쿼리에서 값을 서브쿼리로 전달 (**서브쿼리내에 메인쿼리의 연결 컬럼을 가지고 있음**)
`select a.* from hr.dept a where exists (select deptno from hr.emp x where x.deptno=a.deptno and x.sal < 1300)`
- 단일 행 서브쿼리/다중 행 서브쿼리
 - ✓ 메인 쿼리 – 서브쿼리 연결 연산자가 단순 비교 연산자((=, >, <, >=, <=, <>), in, exists 또는 all/any 인가?)
- in/not in, exists/not exists 차이
- 서브쿼리가 단일 컬럼 또는 다중 컬럼을 반환하는가?

서브쿼리 활용시 유의 사항 정리

비상관 서브쿼리		상관 서브쿼리
단일행 서브쿼리	비교 연산자(=, >, <, >=, <=, <>)	비교 연산자(=, >, <, >=, <=, <>)
다중행 서브쿼리	in, not in	exists, not exists

비상관 서브쿼리에서 IN 연산자 – 다중행 서브쿼리

- 비상관 서브쿼리는 서브쿼리 자체적으로 메인쿼리에 값을 전달
- IN연산자는 괄호내에 여러 개의 상수값 또는 여러 개의 레코드를 반환하는 서브쿼리를 가질 수 있음
- IN연산자는 여러 개의 값이 입력 될 경우 개별 값의 = 조건들의 **OR** 연산이 적용됨
- IN 연산자로 서브 쿼리가 사용될 때 SQL 상에서 중복된 여러 개의 값을 입력하더라도 실제 DB에서 사용될 때는 중복값이 제거된 **유니크한** 값으로 입력됨
- 여러 개의 컬럼 조건으로 IN 연산자를 사용할 수 있음
- **select * from hr.emp where deptno in (20, 30);**
select * from hr.emp where deptno = 20 or deptno=30;
- **select * from hr.dept where deptno in (select deptno from hr.emp where sal < 1300);**
- **select * from hr.dept where (deptno, loc) in (select deptno, 'DALLAS' from hr.emp where sal < 1300);**

비상관 서브쿼리에서 단순 비교 연산자 – 단일행 서브쿼리

- 메인 쿼리에서 단순 비교 연산자(=, >, <, >=, <=, <>)로 서브쿼리의 결과값을 받을 때는 **단 한 개의 행만** 서브쿼리에서 제공해야 함
- 단순 비교 연산자로 서브쿼리를 연결하여도 여러 컬럼 조건을 가질 수 있음.

select * from hr.emp where sal <= (select avg(sal) from hr.emp); **OK**

select * from hr.dept where deptno = (select deptno from hr.emp where sal < 1300); **수행 안됨**

select * from nw.orders
where (customer_id, order_date) = (select customer_id, max(order_date)
from nw.orders where customer_id='VINET' group by customer_id); **OK**

상관 서브쿼리(Correlated Sub-Query)

- 상관 서브쿼리는 서브쿼리내에 메인쿼리의 연결 컬럼을 가지고 있음. 메인쿼리에서 서브쿼리로 연결 값을 전달하는 방식

주문에서 상품 금액이 100보다 큰 주문을 한 주문 정보

```
select * from nw.orders a
where exists (select 1 from nw.order_items x where x.order_id = a.order_id and x.amount > 100);
```

직원의 급여이력에서 가장 최근의 급여이력

```
select * from hr.emp_salary_hist a
where todate = (select max(todate) from hr.emp_salary_hist x where x.empno = a.empno);
```

상관 서브쿼리 exists 연산자 – 다중행 서브쿼리

- Exists는 다중행 상관 서브쿼리에서 사용되는 대표적인 연산자
- 메인쿼리의 레코드 별로 서브쿼리의 결과가 한 건이라도 존재하면 true가 되어 메인쿼리의 결과 반환
- 문맥적으로 메인쿼리의 레코드가 서브쿼리에서 **존재하는지를 체크**하기 위해 활용
- 메인쿼리와 서브쿼리가 단일 컬럼으로 연결시에 IN과 EXISTS의 결과는 서로 동일(물론 서로 호환되는 SQL일 경우만). 하지만 DBMS내부적으로 실행 계획은 서로 다를 수 있음

2건 이상 주문을 한 고객 정보

```
select * from nw.customers a
where exists (select 1 from nw.orders x where x.customer_id = a.customer_id group by customer_id having count(*) >=2);
```

1997년 이후에 한번이라도 주문을 한 고객 정보

```
select * from nw.customers a
where exists (select 1 from nw.orders x where x.customer_id = a.customer_id and x.order_date >= to_date('19970101', 'yyyymmdd'));
```

메인 쿼리와 상관 서브쿼리의 연결 방식

- 메인쿼리는 상관 서브쿼리에서 연결 시 한건의 서브쿼리만 결과를 반환하면 더 이상 동일 레코드로 연결을 수행하지 않기 때문에 메인 쿼리의 집합 레벨을 그대로 유지 할 수 있음.

```
select * from nw.customers a
where exists (select 1 from nw.orders x where x.customer_id = a.customer_id and x.order_date >= to_date('19970101', 'yyyymmdd'));
```



상관 서브쿼리에서 단순 비교 연산자 – 단일행 서브쿼리

- 메인 쿼리에서 상관 서브쿼리의 결과를 단순 비교 연산자(=, >, <, >=, <=, <>)로 비교할 때는 반드시 메인쿼리의 **개별 레코드 별로 단 한 개의 레코드만** 서브쿼리에서 제공해야 함.

직원의 급여이력에서 가장 최근의 급여이력. 메인쿼리의 개별 레코드별로 서브쿼리의 max(todate)는 단 한 개의 레코드를 반환

```
select * from hr.emp_salary_hist a where todate = (select max(todate) from hr.emp_salary_hist x where x.empno = a.empno);
```

아래는 메인쿼리의 개별 레코드 별로 empno 연결조건으로 단 한건이 아닌 여러건을 반환하므로 **수행 오류**

```
select * from hr.emp_salary_hist a where todate = (select todate from hr.emp_salary_hist x where x.empno = a.empno);
```

Null존재 컬럼 서브쿼리 연결 시 not in과 not exists의 차이

- IN연산자는 여러 개의 값이 입력 될 경우 개별 값의 = 조건들의 **OR** 연산이 적용됨
- 하지만 Not IN 조건의 경우 여러개의 개별 = 조건들이 **NOT(조건=) AND NOT(조건=)** 로 변환되어 Null 적용 시 직관과 다른 결과가 발생

```
select * from hr.emp where deptno in (20, 30)
```



```
select * from hr.emp where deptno = 20 or deptno=30
```

부서가 20, 30인 직원 추출

```
select * from hr.emp where deptno in (20, 30, null);
```



```
select * from hr.emp where deptno = 20 or deptno=30 or deptno = null;
```

부서가 20, 30인 직원 추출

```
select * from hr.emp where deptno not in (20, 30, null);
```



```
select * from hr.emp where not (deptno = 20) and not (deptno=30) and not (deptno = null);
```

값이 없음

Null에 연산을 적용 시 Null이 됨. 여러 조건의 결합 시 **True and Null**은 **Null** 이지만 **True or Null**은 **True**가 됨

Null존재 컬럼 서브쿼리 연결 시 not in과 not exists의 차이

- Null 존재 컬럼을 서브쿼리로 연결 시 Not in에서 결과를 추출하려면 서브쿼리 내에서 Null 이 아닌 레코드만 filtering 해야함.
- 이 경우 메인쿼리 레벨에서 서브쿼리 연결 컬럼의 Null값을 가진 데이터는 추출 되지 않음

```
select a.*  
from nw.region a  
where a.region_name not in (select ship_region from nw.orders x where x.ship_region is not null);
```

- Not exists일 경우 별다른 filtering 조건을 추가하지 않아도 직관적으로 수행됨
- 만약 메인쿼리 레벨에서 서브쿼리 연결 컬럼의 Null값을 가진 데이터를 추출하지 않기 위해서는 메인쿼리 레벨에서 is not null로 필터링 해야 함.

```
select a.*  
from nw.region a  
where not exists (select ship_region from nw.orders x where x.ship_region = a.region_name);  
--and a.region_name is not null
```

스칼라 서브쿼리(Scalar Sub-query)

- Select 절에 사용할 수 있는 서브쿼리로 상관 서브쿼리와 유사하게 동작
- 단 한 개의 로우, 단 한 개의 컬럼값만 반환할 수 있음.
- 조인과 유사하게 from절의 집합과 연결되어 원하는 결과를 추출할 수 있으며, **from절의 집합 레벨을 변화 시키지 않음.**
- From 절의 집합에서 한 건씩 스칼라 서브쿼리의 연결되어 값을 반환하므로 **from절 집합이 커질 경우 수행 속도가 저하됨**
- 스칼라 서브쿼리는 Left Outer Join으로 대체 될 수 있음.

```
select a.*,  
       (select dname from hr.dept x where x.deptno=a.deptno) as dname  
from hr.emp a;
```

- 스칼라 서브쿼리는 **편리하지만 남발해서는 안됨**
- 수많은 컬럼들을 스칼라 서브쿼리로 변환하면서 오히려 가독성이 떨어지고, 성능이 저하될 수 있음
- 온라인 쿼리에만 적용.