<Group Function>

그룹함수 중, COUNT(\*)만 Column을 정의하지 않고, Null까지 함께 집계한다.

(역으로 말하자면, Null을 세지 않으려면 반드시 Null을 제외시켜줘야 함)

DISTINCT와 ALL은 그 성질에 맞게, 중복의 제거 유무

SUM(NVL(A, 9))의 경우,

SUM 함수는 본래 NULL을 제외하여 연산하므로 NVL을 사용하더라도 해당 결과값은 전혀 달라지지 않는다. NULL값들에 대한 쓸모없는 NVL 연산을 하게 됨.

단, NVL(Sum(A,0))을 사용하여 NULL로 출력되는 SUM을 0으로 바꿔주는 것은 효율과 무관

(속도 개선에 좋은 요소)

단, AVG(평균값)의 경우, 전체 Data에 대한 평균을 구하는 것을 필요로 하는 때가 많음.

전체 Data에 대한 평균을 구하고 싶은 경우에는, AVG(NVL(A, 0)) 방식으로 전체 평균을 구할 수 있다. (이는 Client / Case에 따라 다름)

(그룹 함수 내에 NVL 함수를 쓰는 거의 유일한 경우)

<Group By>

그룹 함수에 포함되지 않은 SELECT 절의 모든 Column은 반드시 Group By 절에 명시되어야 한다.

SELECT deptno, AVG(sal)

FROM emp

GROUP BY deptno;

집계 함수는 WHERE절에 올 수 없음.

SELECT Column, Group\_Function

FROM table\_name

WHERE condition

GROUP BY group\_by\_expression

HAVING group\_condition

ORDER BY Column

-- 해석 순서 : FROM >> WHERE >> GROUP BY >> HAVING >> SELECT >> ORDER BY

HAVING 절도 GROUP을 만듦.

<HAVING>

GROUP BY에 있는 Column을 활용한 연산식

--> 예제 4의 문장은 효율 면에서 부족함

SELECT deptno, SUM(sal)

FROM emp

GROUP BY deptno

HAVING SUM(sal) > 10000

AND deptno IN (20,30)

가능하면, 일반 Column과 관련된 사항은 WHERE에서 모두 걸러낼 것.

HAVING에서는 정말 GROUP에서 걸러낼 부분을 다루는게 맞다.

그러므로, AND deptno IN(20, 30)은 WHERE절로 뺴자

--> 예제 6

SELECT deptno, AVG(sal)

FROM emp

GROUP BY deptno

HAVING AVG(sal)>1000

AND Job = ‘CLERK’

이는 GROUP에 속하지 않은 JOB Column에 대한 연산을 제시하여 문법 오류가 발생하는 경우.

<LIST>

Group 함수 내 집계된 Data를 확인할 수 있도록 하는 함수.

LISTAGG

LISTAGG는 그룹에 속한 데이터를 불러오는 부분이므로, WITHIN과 함께 쓰여야만 한다.

(에제 9, 예제 11)

<ROLL UP>

계층별 집계가 필요.

Ex.

부서 - 직급별 합계

부서별 합계 (총계)

가독성과 비교성이 올라감.

함수 내 Argument가 나열된 순서대로 계층을 만듦.

>> N + 1가지 종류의 Data가 만들어짐.

ROLLUP (Argu 1, Argu 2, Argu 3)

Argu 3의 합계

Argu 2의 합계

Argu 1의 합계

총계

총 N + 1종류의 Data 생성됨

Cube 함수를 통해 나열된 순서와 상관 없이, 모든 Column

2^n개의 Data 그룹이 만들어짐

Argu 3의 합계

Argu 2의 합계

Argu 1의 합계

Argu 3, 2의 합계

Argu 3, 1의 합계

Argu 2, 1의 합계

Argu 3, 2, 1에 대한 각 합계

전체 총계

ROLLUP의 효율 -> 사용하기에 간결하고, 본래 Data에서 한번만 읽어오므로 성능상 효율성이 높다.

Cube 다차원 집계 – ROLL UP과 달리, 순서에 상관 없이 집계

예제 2 / 예제 3

SELECT deptno, job, SUM(sal)

FROM EMP

GROUP BY ROLLUP(deptno, job);

//

SELECT deptno, job, SUM(sal)

FROM EMP

GROUP BY ROLLUP(job, deptno);

결과가 다른 이유는, 나열하여 만들어내는 순서가 다르기 때문

예제 4

Grouping 함수를 통해, 0 : 실제 Data가 Null / 1 : 가공된 Null을 구분해줘야 한다.

<합성 컬럼>

일부 계층만 제외 또는 일부 계층만 집계

<Grouping Sets>

2개 이상의 Column을 묶은 Argu의 경우에는, 괄호로 묶어서 나열해줘야만 함

<PIVOT - Pivot Table 형태의 Data>

SELECT deptno, sal, decode(deptno, 10, sal)

From emp;

SELECT \*

FROM (SELECT deptno, sal from emp)

PIVOT (SUM(sal) FOR deptno IN (10, 20, 30));

SELECT \*

FROM (SELECT deptno, sal from emp)

PIVOT (sum(sal) for deptno IN (10, 20, 30) ),

(SELECT job, sal from emp)

PIVOT (sum(sal) for job IN(‘ANALYST’, ‘MANAGER’,’CLERK’));

SELECT \*

FROM (SELECT NVL(TO\_CHAR(deptno), ‘합계’) ndeptno,

NVL(job, ‘합계’) njob, SUM(sal) s\_sal

FROM emp

GROUP BY CUBE(deptno, job))

PIVOT (SUM(S\_Sal) FOR ndeptno IN(‘10’, ‘20’, ‘30’, ‘합계’))  
Order BY 1;

**<JOIN>**

다른 Table로부터 Data를 가져와야 하는 경우,

해당 Table의 공통 Column을 통해 Data를 가져오는 것.

ERD를 통해 구조를 보며 익힐 수 있으니, 반드시 ERD를 숙지하는 것이 좋다.

ERD에서는, Primary Key가 실선으로 구분되어 있다. (Table에서 Primary Key는 필수이므로)

FK = Foreign Key

실선가 점선 차이 :

까마귀발 (TR 기호같이 생긴 것)

동그라미 표시 – 동그라미 없는 쪽 Table에 동그라미쪽 Table에 없을 수 있음 (Null 허용 가능)

그러므로, 실선인 경우에는 동그라미가 나타날 수 없음. (NULL이 없으므로)

일치되지 않는 Data에 대해서는 Outer Join을 해야할 경우가 있음.

ERD 읽는 법 – PKFK, 동그라미, 까마귀발, 실선, 점선 등

Oracle : 계층형 쿼리 부분을 직접 작성할 수 있음. 이외 DB에서는 X

<JOIN>

카다시안 곱

SQL 튜닝 목적으로 주로 쓰나, 실수로 발생하기도 함

(117 P)

<Non-Equi Join>

일정 값에 대한 일치 조건이 아닌 범위 조건에 대한 Join을 성립할 때

<Outer Join>

기준 Table을 기준으로 모든 정보를 조회할 때

기준이 아닌 Table에 (+) 부호를 붙여서 사용

ANSI Join – Full Outer Join

Outer Join을 ANSI Join으로 쓰는 경우가 더욱 많음. (예제 4)

(예제 5, 6)

상수 조건에 Outer Join을 붙이지 않으면, Outer Join에 영향을 주게 된다. (상쇄된다)