2장.

2-1. 40~43.

58~61.

4-1. 73~75.

4-2. 76~79.

4-4.

DISTINCT 중복을 제거하고 해당 열을 출력.

4-5.

AS 별칭을 사용. 다른 방법도 있지만 주로 AS + 별칭으로 사용함.

4-6.

ORDER BY 데이터를 정렬된 상태로 출력. SELECT 문에서 가장 마지막에 사용. ASC(오름차순), DESC(내림차순). 사용 안 하면 ASC가 자동 반영됨. 다만, 정렬이 꼭 필요한 경우가 아니면 넣지 않는 게 좋다. 성능을 많이 먹어서 시간이 걸린다.

5-1.

WHERE 데이터 중에서 어떤 조건에 일치하는 행만 조회하는데 사용. WHERE 조건식 = 조회 조건 지정. 결과가 참인 경우에만 출력됨.

5-2.

위에 4챕터에 있던 건 대부분 , 이걸 사용했지만 WHERE에서는 AND, OR을 사용한다(조건식 개수는 사실상 제한이 없다. 여러 개 사용 가능.). 문자열 데이터를 다룰 때는 작음따옴표(‘)로 묶어준다. 공백도 포함한다.

테이블 안에 들어 있는 문자, 문자열 데이터는 대소문자를 구별한다.

5-3.

오라클은 나머지를 구하는 연산자로 mod 함수를 사용한다.

WHERE ENAME >= F;

ENAME의 첫 문자가 F와 같거나 뒤쪽인 것만 검색하는 것을 의미하는 코드.

WHERE ENAME <= ‘FORZ’;

FIND가 ENAME에 있을 경우 첫 문자는 F로 같다. 두 번째 문자 I가 FORZ에서 O보다 알파벳 순서상 빠르므로 조건식에 해당하는 값이 된다. FORD의 경우에 맨 끝 글자 D가 Z보다 앞에 있어서 조건을 만족한다.

!=, <>, ^=

값이 다를 경우에 true를 반환. 같으면 false 반환.

JOB IN (‘A’, ‘B’, ‘C’);

JOB에 있는 a,b,c 중에 하나이면 해당 데이터를 조회한다. OR 문이랑 유사한 기능. 사실상 OR 축약 기능.

NOT의 경우 연산자 앞에 사용한다. 예를 들어 위의 코드에서 JOB이라는 열 이름에 붙이지 않고, JOB NOT IN처럼 연산자 IN 앞에 붙인다.

BETWEEN A AND B는 A이상 B이하의 값의 데이터를 조회.   
  
LIKE 일부 문자열이 포함된 데이터를 조회.

\_,%는 와일드 카드라고 한다. \_는 어떤 값이든 상관없이 한 개의 문자 데이터를 의미. %는 길이와 상관없이 모든 문자 데이터를 의미. LIKE ‘S%’는 시작 문자가 S면 그 뒤의 내용은 상관하지 않고 해당 데이터 조회(%S%는 S를 포함한 데이터 조회). LIKE ‘\_L%’는 두 번째 글자가 L인 데이터 조회(두 번째 글자이므로 L앞에는 어떤 종류와 상관없이 한 문자가 반드시 와야 한다).

ESCAPE.

WHERE SOME\_COLUMN LIKE ‘A\\_A%’ ESCAPE ‘\’;

\ 문자 바로 뒤에 있는 \_는 와일드 카드 기호가 아니라 데이터에 포함된 문자로 인식하라는 의미.

NULL은 현재 무슨 값인지 확정되지 않은 상태 또는 값 자체가 존재하지 않는 상태를 나타냄. 산술 연산자와 비교 연산자로 비교해도 결과 값이 NULL이 된다. WHERE COMM = NULL;로 하면 결과가 아무것도 나오지 않는다. 왜냐하면 WHERE절은 결과가 true인 행만 출력하는데 NULL 때문에 연산 결과 값이 NULL이 돼서 출력 결과에서 제외가 된다.

이때 NULL값을 출력하는 IS NULL이 있다. WHERE COMM IS NULL;은 NULL인 데이터만 출력해준다. IS NULL, IS NOT NULL은 데이터가 NULL인지 아닌지 확인하는 역할을 한다.

OR 연산자는 한쪽의 값이 true이기만 하면 다른 한쪽이 false, NULL 중 어느 거라도 상관없이 값을 출력한다. AND는 false와 NULL 중 false 결과가 우선이고 OR은 false와 NULL 중 NULL이 우선이다.

UNION에서 각 SELECT에서 열 개수, 열 자료형이 순서별로 일치해야함.

UNION 합집합으로 묶음. 중복 제거.

UNION ALL 합집합으로 묶음. 중복 포함.

MINUS 먼저 작성한 SELECT문의 결과 값 중 다음 SELECT문에 존재하지 않는 데이터만 출력.

INTERSECT 교집합. 먼저 작성한 SELECT문과 다음 SELECT문의 결과 값이 같은 데이터만 출력.

6-1.

데이터가 한 행씩 입력되고 한 행당 결과가 하나씩 나오는 건 단일행 함수. 여러 행이 입력되어 하나의 행으로 결과가 반환되면 다중행 함수.

6-2.

INITCAP(문자열) : 첫 글자는 대문자, 나머지는 소문자로 변환.

LENGTHB() : 바이트 수를 반환한다. ‘한글’을 입력하면 4가 나온다. 한글이 1글자당 2바이트이므로.

DUAL 테이블은 더미 테이블. 함수의 결과 값 확인 용도로 사용.

SUBSTR(문자열 데이터, 시작위치, 추출 길이) : 시작위치부터 추출 길이만큼 추출함(추출 길이 없으면 끝까지 출력). 음수면 마지막 위치부터 거슬러 올라간 위치에서 시작.

CLERK가 있을 때 음수가 아니면 C가 1이고 K가 5이다. 맨 왼쪽에서부터 1~5. 여기선 5글자니까. 그런데 음수를 쓰면 C가 –5이고 K가 –1이다. SUBSTR(JOB, -5); 이런 게 들어가면 –5부터 전부 추출하니까 CLERK가 나온다. SUBSTR(JOB, -5, 2); 이렇게 하면 –5에서 2개 추출이니까 CL이 나온다. SUBSTR(JOB, -3); 이렇게 되면 –3인 E부터 전부 출력이니까 ERK가 나온다. 만약 SCLERK라면 S가 –6이다.

INSTR(‘HELLO, ORACLE!’, ‘L’) = 3. L이 3번째에 있음.

INSTR(‘HELLO, ORACLE!’, ‘L’, 5) = 12. L이 12번째에 있음. 5번째인 O에서부터 L을 찾음.

INSTR(‘HELLO, ORACLE!’, ‘L’, 2, 2) = 4. L이 4번째에 있음. 2번째인 E에서부터 2번째 L을 찾음. HELL에서 마지막 L이 4번째.

INSTR은 찾으려는 값이 없으면 0을 반환.

REPLACE(‘010-1234-5678’, ‘-’, ‘ ’) = 010 1234 5678

REPLACE(‘010-1234-5678’, ‘-’) = 01012345678

2번째는 대체할 문자를 입력하지 않아서 찾는 문자인‘-’가 그냥 삭제되었다.

LPAD(‘Oracle’, ‘10’, ‘#’) = ####Oracle

RPAD(‘Oracle’, ‘10’, ‘\*’) = Oracle\*\*\*\*

만약 #이나 \* 등의 값없이 10까지만 쓰고 끝내면 공백이 들어간다. Oracle는 6자리만큼 공간을 차지한다. 이때 지정한 데이터 길이는 10인데, Oracle는 6으로 10보다 작아서 남은 자리를 지정한 문자가 채워주고 있다.

CONCAT(7788, SCOTT) = 7788SCOTT

CONCAT(7788, CONCAT(‘ : ’, SCOTT)) = 7788 : SCOTT

연결시키는 기능. ||도 동일한 기능을 한다.

7788 || SCOTT = 7788SCOTT

TRIM(‘ \_ \_Oracle\_ \_ ’) = [\_ \_Oracle\_ \_] >> 양쪽 공백 제거.

TRIM(LEADING FROM ‘ \_ \_Oracle\_ \_ ’) = [\_ \_Oracle\_ \_ ] >> 왼쪽 공백 제거.TRAILING를 사용하면 오른쪽 공백 제거. BOTH FROM 사용하면 양쪽 공백 제거(맨 위와 같음).

TRIM(‘\_’ FROM ‘\_ \_Oracle\_ \_’) = [ \_Oracle\_ ] >> 양쪽 ‘\_’ 제거.

TRIM(LEADING ‘\_’ FROM ‘\_ \_Oracle\_ \_’) = [ \_Oracle\_ \_] >> 왼쪽 ‘\_’ 제거. TRAILING, BOTH 사용시 위와 같음.

LTRIM(‘ \_Oracle\_ ’) = [\_Oracle\_ ] >> 왼쪽만 제거. RTRIM은 오른쪽만 제거.

LTRIM(‘<\_Oracle\_>’, ‘\_<’) = [Oracle\_>] >> 왼쪽 <\_ 제거. Oracle\_>에서 \_>는 제거 문자인 \_<로는 조합할 수 없어서 안 됨. 애초에 <\_ 다음에 Oracle의 O가 오기 때문에 더 이상 \_< 제거 불가능하기 때문에 삭제 작업이 아예 끝난다. 만약 <\_<\_Oracle 였다면, Oracle만 남는다. RTRIM도 마찬가지로 작동.

6-3.

ROUND(1234.5678) = 1235. 어디서 반올림할지 지정하지 않아서 소수점 첫째자리에서 반올림. ROUND(1234.5678, 0) = 1235. 첫 번째와 같음. ROUND(1234.5678, 1) = 1234.6. 반올림해서 소수점 첫째 자리까지 나타냄. ROUND(1234.5678, -1) = 1230. 1의 자리에서 반올림.

TRUNC는 내림 처리를 하는 것만 다르고 ROUND와 같게 작동.

CEIL(입력된 숫자보다 큰 정수 중 가장 작은 수), FLOOR(입력된 숫자보다 작은 정수 중 가장 큰 수).

MOD(15. 6) = 3. 15를 6으로 나누고 나머지 값을 출력.

6-4.

날짜 데이터 + 날짜 데이터 = 연산 불가. 지원 안 함. -는 가능. 숫자는 +, - 다 가능.

SYSDATE-1 = 현재 날짜 – 1. 현재 날짜가 2018-07-13일 경우 이 식의 값은 2018-07-12가 된다.

ADD\_MONTHS(SYSDATE, 3) = 현재 날짜가 2018-07-13이면 값은 2018-10-13. 10년을 더하는 거면 120. 이런 식으로 작성 가능.

MONTHS\_BETWEEN(HIREDATE, SYSDATE) = -450.12345678. 이런 식으로 값이 나온다. 먼저 입력한 HIREDATE – SYSDATE 개념. 만약 위에서 위치가 반대면 양수가 나온다.

SYSDATE는 위에 값을 기준으로 사용.

NEXT\_DAY(SYSDATE, ‘월요일’) = 2018-07-16. 13일에서 다음 월요일이 오는 날 계산.

LAST\_DAY(SYSDATE) = 2018-07-31. 해당 달이 속한 달의 마지막 날짜를 반환.

ROUND, TRUNC는 날짜 데이터에도 사용 가능. 반올림 위치 자리에 반올림 기준 포맷을 사용한다.

ROUND(SYSDATE, ‘CC’) = 2001-01-01.

6-5.

숫자데이터 + ‘500’ 이런 식으로 하면 문자 데이터 ‘500’이 자동으로 숫자로 형 변환이 된다. 그러나 반대로 ‘ABCD’ + 문자데이터 이것의 경우엔 자동 형 변환이 되지 않는다.

TO\_CHAR(SYSDATE, ‘YYYY/MM/DD HH24:MI:SS’) = 2018/07/13 23:59:01. 이런 식으로 나온다. TO\_CHAR(SYSDATE, ‘MM’) = 07. 이런 식으로 형식을 지정해주면 됨.

TO\_CHAR(SYSDATE, ‘MON’, NLS\_DATE\_LANGUAGE = ENGLISH’) = JUL. 다른 언어로 변경 가능.

TO\_CHAR(SYSDATE, ‘HH24:MI:SS’) = 24시간으로 표현한 시간. HH12는 12시간으로 표현한 시간. AM, A.M., PM, P.M. 오전, 오후 표시.

SAL은 800이라고 했을 때.

TO\_CHAR(SAL, ‘$999,999’) = $800.

TO\_CHAR(SAL, ‘L999,999’) = \800.

TO\_CHAR(SAL, ‘999,999.00’) = 800.00

TO\_CHAR(SAL, ‘000,999,999.00’) = 000,000,800.00

TO\_CHAR(SAL, ‘000999999.99’) = 000000800.00

TO\_CHAR(SAL, ‘999,999,00’) = 8,00

TO\_NUMBER(‘1,300’, ‘999,999’) - TO\_NUMBER(‘1,500’, ‘999,999’) = -200. 문자열 안에 있는 숫자에 ‘,’ 여기에 , 이게 있어서 숫자로 형 변환을 해줘야함. 자동 형 변환이 안 됨.

TO\_DATE(‘2018-07-14’, ‘YYYY-MM-DD’) = 2018/07/14. 처음 부분에 ‘20180714’로 해도 같은 결과 나옴. 날짜 데이터는 상대적으로 이전 날짜인 데이터가 이후 날짜 데이터보다 크기가 작은 데이터로 여겨지고 비교 연산자도 사용 가능.

연도 표시에서 YY는 현 시점의연도와 동일한 연도로 계산. RR은 현 시점의연도의 끝 두 자리 수가 00~49, 50~99 그리고 입력된 수치가 00~49, 50~99인 경우를 계산해서 비교적 가까운 날짜 데이터를 계산한다.

TO\_DATE(‘49/12/10’, ‘YY/MM/DD’) = 2049/12/10. RR도 동일. 그런데 50/12/10은 YY에서는 동일하게 작동하지만 RR에서는 1950/12/10을 반환. 51의 경우에도 RR은 1951/12/10을 반환한다.

6-6.

SELECT EMPNO, NVL(COMM, 0) = COMM이 NULL이 아니면 해당 데이터를 그대로 반환. NULL이면 0을 반환.

NVL2(COMM, ‘O’, ‘X’) = NULL이 아닐 경우 O 반환. NULL일 경우 X반환.

6-7.

SELECT EMPNO, DECODE(JOB, ‘MANAGER’, SAL\*1.1, ‘SALESMAN’, SAL\*1.05, SAL\*1.03) = 직책이 매니저면 급여 10%인상. 판매원이면 급여 5% 인상. 그 외의 나머지는 급여 3% 인상. 반환 값 지정 안 하면 NULL 반환.

SELECT EMPNO, CASE JOB

WHEN ‘MANAGER’ THEN SAL\*1.1

WHEN ‘SALESMAN’ THEN SAL\*1.05

ELSE SAL\*1.03

END AS TEST

이건 CASE문으로 위와 같음. 추가로 CASE에는 DECODE와 달리 JOB 같이 기준 데이터를 지정하지 않아도 된다.

SELECT EMPNO, CASE

WHEN COMM IS NULL THEN ‘해당사항없음’ >> COMM 값이 NULL이면 해당사항없음 출력.

WHEN COMM > 0 THEN ‘수당 : ’ || COMM >> COMM이 0초과이면 수당 : COMM 값 출력.

END AS TEST

DECODE, CASE 모두 조건별로 동일한 자료형의 데이터를 반환해야 한다.

7-1.

SELECT SUM(SAL) = SAL 열에 있는 모든 값의 합이 하나의 행에서 출력된다. 다중행 함수를 사용한 SELECT에서는 여러 행이 결과로 나올 수 있는 열은 함께 사용 불가능. SELECT ENAME, SUM(SAL)은 불가능. ENAME은 해당 열에 저장된 이름을 하나씩 출력한다. 그런데 SUM(SAL)은 SAL의 합계가 하나만 나온다. 덧셈으로만 합계를 구할 때 NULL이 있으면 결과는 무조건 NULL이 나오지만 SUM 함수는 NULL을 제외하고 합을 구해서 NULL이 열에 있어도 상관없다.

앞에 DISTINCT, ALL을 사용할 수 있다. SUM(DISTINCT, SAL) 이런 식. DISTINCT는 중복 값은 한 번만 합계에 사용하고 그 이후론 제외한다. ALL은 사용하지 않았을 때와 같다.

COUNT 함수에 \*을 사용하면 SELECT문의 결과 값으로 나온 행 데이터의 개수를 반환함. NULL 값은 반환 개수에서 제외된다. 세지 않는다. 값에 1, 2, NULL이 있으면 COUNT 값은 2가 나옴.

MAX(SAL), MIN(SAL) 각각 최댓값과 최솟값을 반환. 숫자, 문자, 날짜 모두 가능. 날짜는 예전 날짜보다 최근 날짜가 더 큰 값.

AVG(SAL) 평균을 구함.

7-2.

GROUP BY에서는 AS로 만든 별칭이 인식되지 않는다. 열 이름이나 연산식을 그대로 지정해야함.

SELECT AVG(SAL), DEPTNO

FROM EMP

GROUP BY DEPTNO;

이렇게 하면 각 부서별 평균 급여를 볼 수 있다.

위를 변형해서

SELECT DEPTNO, JOB, AVG(SAL)

FROM EMP

GROUP BY DEPTNO, JOB;

각 부서의 직책별 평균 급여가 나온다. GROUP BY에서 명시된 부서 번호로 그룹을 먼저 묶고, 그룹내에서 사원 직책 열을 기준으로 다시 소그룹을 묶어 평균을 출력한다.

7-3.

HAVING은 GROUP BY가 존재할 때만 사용 가능. GROUP BY 아래에 HAVING AVG(SAL) >= 2000 등으로 WHERE처럼 사용. 별칭 사용 불가. WHERE는 출력 대상 행을 제한. HAVING는 그룹화된 대상을 출력에서 제한.

7-4.

ROLLUP함수에는 그룹 함수 지정 불가. 예를 들어 EMPNO, ENAME, SAL 등은 넣을 수 있지만 SUM(SAL), AVG(SAL) 등은 넣을 수 없다는 뜻.

ROLLUP(A, B, C) =

1. A 그룹별,B 그룹별, C그룹별 결과 출력.

2. A 그룹별,B 그룹별 결과 출력.

3. A 그룹 결과 출력.

4. 전체 데이터 출력.

ROLLUP안에 넣은 값의 +1 조합이 출력이 된다. 여기서 1은 GROUP BY만 사용했을 때와 동일한 결과다. 그런데 2,3,4이 추가되어 있다. 따라서 각 그룹의 합계가 따로 출력이 된다.

CUBE(A, B, C)

1. A 그룹별,B 그룹별, C그룹별 결과 출력.

2. A 그룹별,B 그룹별 결과 출력.

3. B 그룹별,C 그룹별 결과 출력.

4. A 그룹별,C 그룹별 결과 출력.

5. A 그룹 결과 출력.

6. B 그룹 결과 출력.

7. C 그룹 결과 출력.

8. 전체 데이터 출력.

CUBE는 모든 조합을 출력한다.

SELECT DEPTNO, JOB, COUNT(\*)

FROM EMP

GROUP BY DEPTNO, ROLLUP(JOB)

ORDER BY DEPTNO;

이 코드의 경우 그룹에서 DEPTNO와 ROLLUP(JOB)로 그룹을 만든다. 그래서 번호와 직책으로 묶는다. 이렇게만 그냥 그룹으로 위의 DEPTNO, JOB를 한 것과 마찬가지다. 하지만 JOB는 ROLLUP에 들어가 있으므로, JOB그룹과 전체 데이터 출력 2개를 실행하므로 JOB 1개가 아니라 2개라고 봐도 된다. DEPTNO도 여기에 맞춰서 JOB와 1대1로 그룹으로 묶고, 그 다음엔 JOB의 합계 데이터를 출력하는데 이를 DEPTNO도 그룹으로 묶었으니 각 그룹 번호(여기서는 부서 번호)의 합계가 출력이 된다.

GROUPING SETS = 지정한 모든 열을 각각 대그룹으로 처리해서 출력.

GROUPING = ROLLUP, CUBE 함수를 사용한 GROUP BY영역에서 지정한 열이 그룹화된 상태로 결과가 집계되었는지 확인하는 용도. 0,1로 출력됨. 0은 그룹핑 함수에 지정한 열이 그룹화되었음을 의미하고, 1은 그룹화되지 않은 데이터를 의미한다.

GROUPING\_ID = GROUPING와 같은 기능. 다만 GROUPING는 검사할 열을 하나씩 지정했다면 이건 한 번에 여러 열 지정 가능.

|  |  |  |
| --- | --- | --- |
| 그룹화 된열 | 그룹화 비트 벡터 | 결과 |
| a,b | 0 0 | 0 |
| a | 0 1 | 1 |
| b | 1 0 | 2 |
| 없음 | 1 1 | 3 |

01의 값을 2진수로 보고 10진수로 바꾼 값이 최종 값으로 출력됨. 위에서도 11=3, 10=2.

LISTAGG(ENAME, ‘, ’) WITHIN GROUP(ORDER BY SAL DESC) = 사원 이름 가로로 출력.

PIVOT(MAX(SAL) FOR DEPTNO IN (10, 20, 30)) = 행을 열로 바꿈. 가로줄로 표시할 열을 FOR로 명시하고 IN여기에 출력하려는 열 데이터 지정. 이 데이터에 별칭 설정 가능.

SELECT DEPTNO, JOB, MAX(SAL)

FROM EMP

GROUP BY DEPTNO, JOB

ORDER BY DEPTNO, JOB;

SELECT \*

FROM(SELECT DEPTNO, JOB, SAL FROM EMP)

PIVOT(MAX(SAL) FOR DEPTNO IN (10, 20, 30))

ORDER BY JOB;

SELECT \*

FROM(SELECT JOB, DEPTNO, SAL FROM EMP)

PIVOT(MAX(SAL) FOR JOB IN ('CLERK' AS CLERK,'SALESMAN' AS SALESMAN,

'PRESIDENT' AS PRESIDENT, 'MANAGER' AS MANAGER, 'ANALYST' AS ANALYST))

ORDER BY DEPTNO;

UNPIVOT는 위와 반대 기능.

SELECT

DEPTNO,

MAX(DECODE(JOB, 'CLERK', SAL)) AS CLERK,

MAX(DECODE(JOB, 'SALESMAN', SAL)) AS SALESMAN,

MAX(DECODE(JOB, 'PRESIDENT', SAL)) AS PRESIDENT,

MAX(DECODE(JOB, 'MANAGER', SAL)) AS MANAGER,

MAX(DECODE(JOB, 'ANALYST', SAL)) AS ANALYST

FROM EMP

GROUP BY DEPTNO

ORDER BY DEPTNO;

SELECT \*

FROM(SELECT DEPTNO,

MAX(DECODE(JOB, 'CLERK', SAL)) AS CLERK,

MAX(DECODE(JOB, 'SALESMAN', SAL)) AS SALESMAN,

MAX(DECODE(JOB, 'PRESIDENT', SAL)) AS PRESIDENT,

MAX(DECODE(JOB, 'MANAGER', SAL)) AS MANAGER,

MAX(DECODE(JOB, 'ANALYST', SAL)) AS ANALYST

FROM EMP

GROUP BY DEPTNO

ORDER BY DEPTNO)

UNPIVOT(SAL FOR JOB IN (CLERK, SALESMAN, PRESIDENT, MANAGER, ANALYST))

ORDER BY DEPTNO, JOB;