SQL 명령어

SQL – Structured Query Language

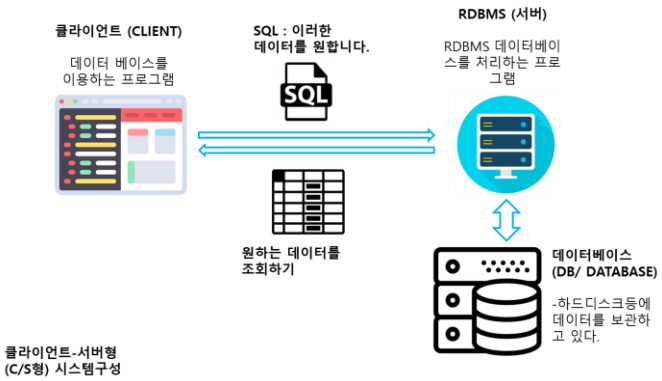
DML - SELECT, INSERT, UPDATE, DELETE
DDL - CREATE, ALTER, DROP, RENAME
DCL - GRANT, REVOKE
TCL - COMMIT, ROLLBACK

TABLE (행, 열, 컬럼)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	7902	80/12/17	800
7499	ALLEN	SALESMAN	7698	81/02/20	1600
7521	WARD	SALESMAN	7698	81/02/22	1250
7566	JONES	MANAGER	7839	81/04/02	2975
7654	MARTIN	SALESMAN	7698	81/09/28	1250
7698	BLAKE	MANAGER	7839	81/05/01	2850
7782	CLARK	MANAGER	7839	81/06/09	2450
7788	SCOTT	ANALYST	7566	87/04/19	3000
7839	KING	PRESIDENT	(null)	81/11/17	5000
7844	TURNER	SALESMAN	7698	81/09/08	1500
7876	ADAMS	CLERK	7788	87/05/23	1100
7900	JAMES	CLERK	7698	81/12/03	950
7902	FORD	ANALYST	7566	81/12/03	3000
7934	MILLER	CLERK	7782	82/01/23	1300

DBMS의 구조

CLIENT – SERVER 시스템



클라이언트-서버형 (C/S형) 시스템구성

OBJECT

DBMS(Database Management System, 데이터베이스 관리 시스템)에서 "오브젝트"는 데이터베이스 내에 존재하는 다양한 유형의 항목

데이터의 유형

- CHAR(s) : 고정 길이 문자열 정보.
- VARCHAR2(s) : 가변 길이 문자열 정보.
- NUMBER : 정수, 실수 등 숫자 정보
- DATE : 날짜와 시각 정보

SELECT

SELECT 문장을 사용함으로 , 원하는 변수와 원하는 행(ROW)을 선택적으로 조회가 가능하다.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	7902	80/12/17	800
7499	ALLEN	SALESMAN	7698	81/02/20	1600
7521	WARD	SALESMAN	7698	81/02/22	1250
7566	JONES	MANAGER	7839	81/04/02	2975
7654	MARTIN	SALESMAN	7698	81/09/28	1250
7698	BLAKE	MANAGER	7839	81/05/01	2850
7782	CLARK	MANAGER	7839	81/06/09	2450
7788	SCOTT	ANALYST	7566	87/04/19	3000
7839	KING	PRESIDENT	(null)	81/11/17	5000
7844	TURNER	SALESMAN	7698	81/09/08	1500
7876	ADAMS	CLERK	7788	87/05/23	1100
7900	JAMES	CLERK	7698	81/12/03	950
7902	FORD	ANALYST	7566	81/12/03	3000
7934	MILLER	CLERK	7782	82/01/23	1300

SELECT * FROM EMP;

SELECT EMPNO, ENAME FROM EMP;

CONCAT

[ORACLE]

<문자열 혹은 컬럼명> || <문자열 혹은 컬럼명>

[SQL SERVER]

<문자열 혹은 컬럼명> + <문자열 혹은 컬럼명>

CONCAT(<문자열 혹은 컬럼명>, <문자열 혹은 컬럼명>)

SELECT 'ABC' || 'DEF' FROM EMP; : ABCDEF

DISTINCT

DISTINCT 연산자는 로우 정보들 중에서 중복되는 값을 제거하여 하나만 남기는 집약 기능을 한다.

실행 예

►SELECT DISTINCT DEPTNO FROM EMP;

DEPTNO
10
20
30

실행 예

►SELECT DEPTNO FROM EMP;

DEPTNO
20
30
30
20
30
30
10
20
10
30
20
30
20
10

DISTINCT - 두 개 이상의 인수를 사용하는 경우

실행 예

►SELECT DEPTNO,JOB FROM EMP;

DEPTNO	JOB
20	CLERK
30	SALESMAN
30	SALESMAN
20	MANAGER
30	SALESMAN
30	MANAGER
10	MANAGER
20	ANALYST
10	PRESIDENT
30	SALESMAN
20	CLERK
30	CLERK
20	ANALYST
10	CLERK

실행 예

►SELECT DISTINCT DEPTNO,JOB FROM EMP;

DEPTNO	JOB
20	CLERK
30	SALESMAN
20	MANAGER
30	CLERK
10	PRESIDENT
30	MANAGER
10	CLERK
10	MANAGER
20	ANALYST

ALIAS 연산자의 사용

SELECT *<컬럼명>* AS *<Alias 컬럼명>*

SELECT *<컬럼명>* AS *"<Alias 컬럼명>"*

SELECT *<컬럼명> <Alias 컬럼명>*

SELECT *<컬럼명> "<Alias 컬럼명>"*

※ ALIAS 는 COLUMN HEADER 을 변경하는 기능을 한다. (기출)

ALIAS – 공백을 포함한 별칭

- " (쌍따옴표) 는 문자열이 공백을 포함하는 경우 사용된다.
- 공백이 없으면 그냥 따옴표 없이도 사용 가능하다.
- " (쌍따옴표) 는 문자열이 특수문자를 포함하는 경우에도 사용된다.

WHERE

WHERE 절은 테이블의 각각의 행(개체)에 대해서 개체 조건을 만족하는지 여부를 판단한다. 해당 행이 조건을 만족시키면 출력한다. 조건이 '참(TRUE)' 이면 출력하고 거짓(FALSE)' 이면 출력하지 않는다.

SAL 이 2000 이 상인가?

ENAME	SAL	
SMITH	800	FALSE
ALLEN	1600	FALSE
WARD	1250	FALSE
JONES	2975	TRUE
MARTIN	1250	FALSE
BLAKE	2850	TRUE
CLARK	2450	TRUE
SCOTT	3000	TRUE
KING	5000	TRUE
TURNER	1500	FALSE
ADAMS	1100	FALSE
JAMES	950	FALSE
FORD	3000	TRUE
MILLER	1300	FALSE

SAL 이 2000 이 상인가?

ENAME	SAL
JONES	2975
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
FORD	3000

비교연산자 : < > <= >= =
같지 않다 : <> , !=, ^=

논리연산자
<조건식1> AND <조건식2>
좌우 조건식이 모두 참일 경우 참
<조건식1> OR <조건식2>
좌우 조건식중 하나만 참일 경우 참
NOT <조건식>
조건식의 결과를 부정

연산자 우선순위

1. =, !=, <, >, <=, >=
2. IS NULL, LIKE, BETWEEN, IN, EXISTS
3. NOT
4. AND
5. OR

※위에 있는 연산자일수록 먼저 연산이 이루어진다.

SQL 연산자

BETWEEN *<최소값>* AND *<최대값>* : 범위 조건
IN*(<값1>, <값2>, ...*): 여러 개 값들 중 같은 것이 있는 경우 참
<값1> IN (<값2>, <값3>, <값4>) 의 의미 :
<값1> = <값2> OR <값1> = <값3> OR <값1> = <값4>
<값1> NOT IN (<값2>, <값3>, <값4>)의 의미 :
<값1> ≠ <값2> AND <값1> ≠ <값3> AND <값1> ≠ <값4>

LIKE 연산자

문자열 칼럼에 저장되어 있는 값이 특정 문자열을 포함하고 있는지 볼 때, like를 사용한다.

'_' : 특정 미지의 글자 하나를 의미한다.
'_ A_' : (아무거나 1글자 + A + 아무거나1 글자)
앞뒤로 어떤 글자가 와도 상관 없지만 개수는 같다.
예) '_ A_': "WAY", "MAY", ...

'%' : 미지의 글자 0개 이상을 의미한다.
'%A%' : 앞뒤로 어떤 글자가 와도 상관없고 몇 글자가 와도 상관이 없다. A 문자열이 포함되어 있으면 참이다.
예) 'H%': " H", " He", " Hi", " Hello"

ESCAPE 연산자

찾고자 하는 와일드카드에 '_' , '%' 가 포함된 문자열을 지정하는 경우 사용

ENAME LIKE 'A_#%' EXCAPE '#'
: '#' 기호 앞의 '_' 는 와일드카드가 아니라 특수문자이다.

TOP-N 쿼리 (가상컬럼)

하나의 테이블에 존재하는 다른 컬럼 값들을 이용해서 만들어진 임시 컬럼
ORACLE에서 행의 번호를 나타내는 가상 컬럼은 ROWNUM
SQL SERVER에서 상위 행을 출력하는 함수는 TOP 함수

SELECT *<컬럼명>*
FROM *<테이블명>*
WHERE *<ROWNUM 조건식>*

SELECT TOP (*<반환할 행의 숫자>***)** *<컬럼 리스트>*
FROM *<테이블명>* **;**

WHERE 조건문의 ROWNUM 은 반드시 ROWNUM=1 인 값을 포함해야 하고 목표값까지 1씩 증가해야 한다.

SELECT EMPNO,ENAME
FROM EMP
WHERE ROWNUM>1; -> 오류 발생

TOP (<반환할 행의 숫자>)*<컬럼 리스트>* **WITH TIES**
※WITH TIES : 동일한 데이터가 있을 경우 함께 출력된다. 사용하기 위해서는 ORDER BY 절이 반드시 필요하다. ORDER BY 절 뒤의 컬럼이 동일한 데이터 여부를 판단하는 기준이다.

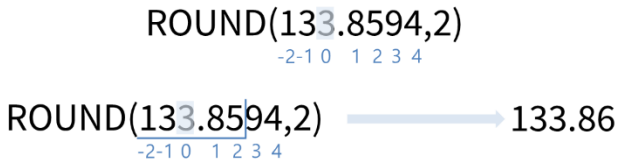
단일행 함수
SQL에서 기본 제공되는 함수를 내장 함수(Built – in Function)라 한다. 단일 행 함수는 하나의 행을 입력했을 때, 결과가 하나의 행으로 나오는 것이다.

DUAL 테이블
오라클 설치시 자동적으로 생성되는 오라클의 표준 테이블
오직 하나의 행(Row)에 하나의 컬럼만 가지고 있는 Dummy 테이블
Dual 테이블은 일시적인 가상테이블로서 숫자 연산, 날짜 연산을 위해서 쓰이는 것이 일반적

단일행 숫자 함수

► **ABS** *(<숫자>)* **: 절대값을 구해주는 함수이다.**
► **SIGN** *(<숫자>)* **: 부호를 출력해주는 함수이다.**
► **FLOOR** *(<숫자>)* **: 소수점을 모두 버리는 함수이다.**
► **CEIL** *(<숫자>)* **: 소수점을 모두 올리는 함수이다.**
► **(※ SQL SEVER) CEILING** *(<숫자>)*
► **MOD** *(<숫자1>, <숫자2>)* **: <숫자1>/<숫자2>의 나머지를 구하는 함수이다.**
► **ROUND** *(<숫자>,<자리수>)* **: 반올림 함수이다.**
► **TRUNC** *(<숫자>,<자리수>)* **: 내림 함수이다.**
※ 자리수에 아무것도 쓰지 않으면 기본값은 0이다.

ROUND 함수



단일행 문자열 함수

► LOWER *(<문자열>)* : 대문자를 소문자로 바꾸는 함수
► UPPER *(<문자열>)* : 소문자를 대문자로 바꾸는 함수
► INITCAP *(<문자열>)* : 첫 글자를 대문자로, 나머지를 소문자로 바꾸는 함수
► CONCAT *(<문자열1>,<문자열2>)* : 두 개의 문자열을 합쳐서 출력해주는 함수
► LENGTH *(<문자열>)* : 문자의 개수를 출력해주는 함수
► LENGTHB *(<문자열>)* : 문자의 바이트를 출력해주는 함수
► SUBSTR *(<문자열>,<숫자>)* : *<숫자>*번째 글자를 포함하여 이후의 문자열을 가져온다.
► SUBSTR *(<문자열>,<숫자1>,<숫자2>)* : *<숫자1>*번째 글자를 포함하여 *<숫자1>*부터 *<숫자2>*까지의 문자열을 가져온다.
► LPAD *(<문자열1>,<숫자>,<문자열2>)*: *<문자열1>* 을 출력한 뒤, 남은 *<숫자>* 바이트의 문자열만큼 좌측으로 *<문자열2>*을 추가한다.
► RPAD *(<문자열1>,<숫자>,<문자열2>)*: *<문자열1>* 을 출력한 뒤, 남은 *<숫자>* 바이트의 문자열만큼 우측으로 *<문자열2>*을 추가한다.
► LTRIM*(<문자열1>, <문자열2>)*: *<문자열1>* 좌측에서부터 *<문자열2>*가 나타나면 다른 문자가 나올 때까지 제거한다.
► RTRIM*(<문자열1>, <문자열2>)*: *<문자열1>* 우측에서부터 *<문자열2>*가 나타나면 다른 문자가 나올 때까지 제거한다.

단일행 날짜 함수

ORACLE 예

```
►SELECT SYSDATE FROM DUAL;
```

SQL SERVER 예

```
► SELECT GETDATE();
```

ORACLE 예

```
►SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL;
►SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;
►SELECT EXTRACT(DAY FROM SYSDATE) FROM DUAL;
```

SQL SERVER 예

```
► SELECT DATEPART(YEAR, GETDATE());
► SELECT DATEPART(MONTH, GETDATE());
► SELECT DATEPART(DAY, GETDATE());
```

- 날짜 연산 (X일 이후 / 이전)

ORACLE 예

```
►SELECT SYSDATE+10 FROM DUAL;
```

SQL SERVER 예

```
► SELECT DATEADD(DAY, 10, GETDATE());
```

- SYSDATE 이외에도 컬럼명으로도 연산을 할 수 있다.

명령문 예

```
►SELECT HIREDATE, HIREDATE+100 FROM EMP;
```

연산	설명
날짜 + 숫자	날짜에 숫자 만큼의 날짜를 더한다. 날짜 데이터로 출력된다.
날짜 – 숫자	날짜에 숫자 만큼의 날짜를 뺀다. 날짜 데이터로 출력된다.
날짜1 – 날짜2	날짜1 에서 날짜2 를 뺀 일수를 구한다. 숫자 데이터로 출력된다.

형변환 함수

<ORACLE>

변환형 함수 - Oracle	설명
TO_NUMBER(<문자열>)	<문자>를 숫자 데이터로 변환한다.
TO_CHAR(<숫자>)	<숫자>를 문자 데이터로 변환한다.
TO_CHAR(<날짜>,<형식>)	<날짜>를 주어진 <형식> 에 따른 형태의 문자열 타입으로 변환한다.
TO_DATE(<문자열>,<형식>)	<문자열>을 주어진 <형식> 에 따른 형태의 날짜 타입으로 변환한다.

CASE 함수

► CASE WHEN *<조건식1>* THEN *<반환값1>*
WHEN *<조건식2>* THEN *<반환값2>* ...
END

<조건식1> 에 만족하면, <반환값1> 을 출력하고, <조건식2>에 만족하면, <반환값2>을 출력한다.

CASE WHEN *<조건식1>* THEN *<반환값1>*
WHEN *<조건식2>* THEN *<반환값2>* ...
ELSE
END

ELSE 절을 사용하는 경우 :
<조건식1> 에 맞는 경우, <조건식2>에 맞는 경우를 제외한 경우 ELSE 의 <반환값> 이 반환된다.
ELSE 절을 사용하지 않는 경우 :
<조건식1> 에 맞는 경우, <조건식2>에 맞는 경우를 제외한 경우 NULL 이 반환된다.

DECODE

► DECODE (*<컬럼명>*, *<값1>*, *<반환값1>*,
***<값2>*, *<반환값2>*,**
***<값3>*, *<반환값3>*, ...)**

<컬럼명>이 <값1>이면 <반환값1>을 가져오고, <값2>이면 <반환값2>를 가져오고 <값3>이면 <반환값3>를 가져온다.

<컬럼명> 이 <값1>, <값2>, <값3> 모두 같지 않는 경우에는 NULL 이 반환된다.

NULL

데이터의 값이 알려져 있지 않거나 의미가 없는 경우에 NULL을 사용한다. 즉, NULL은 값의 부재 혹은 모르는 값

NULL 산술연산

NULL 과의 산술연산은 모두 NULL로 출력된다.
►SELECT NULL+2, NULL-2, NULL/2, 2/NULL FROM DUAL;

NULL과의 비교연산 – IS NULL / IS NOT NULL
►SELECT EMPNO, COMM FROM EMP WHERE COMM IS NULL;
►SELECT EMPNO, COMM FROM EMP WHERE COMM IS NOT NULL;

NULL 관련 함수
- NVL(값1,값2) : 값1의 값이 NULL 이면 값2 출력.
- NULLIF(값1,값2) : 값1이 값2와 같으면 NULL을 아니면 값1을 출력
- COALESCE(값1,값2) : NULL이 아닌 최초의 표현식, 모두 NULL이면 NULL 반환
e.g. COALESCE(NULL, NULL, 'abc') -> 'abc'

NULL의 특징

1. 기본적인 의미는 값의 부재, 모르는 값을 의미한다.
2. 정렬에서는 무한대의 의미를 가진다. (Oracle)
3. 정렬에서는 최소의 값이라는 의미를 가진다. (SQL Server)
4. NULL/2 , 2/NULL, NULL+NULL, NULL-2 모두 NULL값으로 출력된다.
5. (SELECT 절에서) NULL = 3 등의 비교연산 시, UNKNOWN(알수 없음) 이 반환되어 오류가 발생한다. (WHERE 절에서) NULL = 3 등의 비교연산 시, UNKNOWN(알수 없음)이 조건절(WHERE)에 들어가서 거짓(FALSE)의 결과와 같은 결과가 반환된다.

ORDER BY

SELECT 문을 통해 얻어온 결과를 특정 컬럼을 기준으로 오름차순 혹은 내림차순으로 정렬할 수 있다.WHERE, ROWNUM 등과 함께 쓸 수 있다.
숫자, 문자열, 날짜 등 모든 타입의 데이터를 정렬할 수 있다.
ORDER BY 의 정렬 시점은 모든 실행이 끝난 이후, 데이터 출력 전이다.
ORDER BY 는 데이터 베이스의 메모리를 많이 사용한다. (성능 저하의 요인이 된다.)

► **SELECT** ***<컬럼명>*** **FROM** ***<테이블명>***
ORDER BY <컬럼명> **[ASC | DESC];**

ASC : 오름차순 (생략가능),
DESC : 내림차순

컬럼 번호를 이용한 정렬

►**SELECT EMPNO, ENAME, SAL FROM EMP**
ORDER BY 3 ASC;

출력되는 결과의 컬럼의 번호를 기준으로 정렬하는 것이 가능하다.
하지만, 출력되는 결과의 컬럼 숫자보다 큰 값을 이용하여 정렬하면 오류가 발생한다.

출력 되지 않는 컬럼을 이용한 정렬

►**SELECT EMPNO,ENAME,DEPTNO**
FROM EMP
ORDER BY SAL DESC;

복수 컬럼을 이용한 정렬

►**SELECT ENAME, SAL,COMM FROM EMP**
ORDER BY SAL DESC, ENAME ASC;
먼저 SAL 내림차순으로 정렬하고
SAL가 같다면 ENAME 오름차순으로 정렬한다.

정렬의 특징

· 숫자, 문자열, 날짜 등 모든 타입의 데이터를 정렬할 수 있다.
· ORDER BY 의 정렬 시점은 모든 조회(SELECT)가 끝난 이후이다
· ORDER BY 는 데이터 베이스의 메모리를 많이 사용한다. (데이터 조회 성능 저하의 요인이 된다.)

집계함수

► **SUM** ***(<컬럼명>)*** **: 컬럼 값들의 총합을 구한다**
► **AVG (*<컬럼명>) :* 컬럼 값들의 평균을 구한다**
► **COUNT (*<컬럼명>) :* 컬럼 값들의 총 개수을 구한다**
► **MAX (*<컬럼명>) :* 컬럼 값들의 최대값을 구한다**
► **MIN (*<컬럼명>) :* 컬럼 값들의 최소값을 구한다**

집계함수와 NULL의 관계

(원칙) : 집계 함수는 NULL을 제외하고 연산하는 것이 원칙이다.

(예외) : COUNT(*) 는 행의 수를 센다.

COL1	COL2	COL3	COL4
1	1	1	1
NULL	1	NULL	NULL
3	NULL	3	3
NULL	4	NULL	4

<Q17>

① SELECT SUM(COL1+COL2+COL3+COL4)
FROM Q17
② SELECT SUM(COL1)+SUM(COL2)+
SUM(COL3)+SUM(COL4)
FROM Q17

집계함수 WHERE 절에 사용 불가

집계함수는 WHERE 절에 올 수 없다. (집계 함수를 사용할 수 있는 GROUP BY 절보다 WHERE절이 먼저 수행된다.)

►SELECT EMPNO,ENAME, SAL
FROM EMP
WHERE SAL>AVG(SAL) [오류발생]

집계함수 ORDER BY 절 사용 가능

► SELECT DEPTNO, MAX(SAL)
FROM EMP
GROUP BY DEPTNO
ORDER BY MAX(SAL);

GROUP BY

데이터들을 기준컬럼에 따라서 원하는 그룹으로 나눌 수 있다. 개체 속성들은 GROUP BY를 기점으로 그룹수준의 속성들이 된다.

► GROUP BY *<컬럼명>*
► SELECT
FROM
WHERE
GROUP BY *<컬럼명>*

HAVING
그룹수준 속성들의 조건절에 해당

GROUP BY NULL

► SELECT SUM(SAL)
FROM EMP
GROUP BY NULL;

► SELECT SUM(SAL)
FROM EMP;

그룹함수

제품분류	SUM(매출합계)
사무용품	19466000
지류	28039000
화방용품	72386500

```
SELECT 제품분류,SUM(매출합계)
FROM 문방구
GROUP BY 제품분류;
```

ROLLUP (인수 한 개)

제품분류	SUM(매출합계)
사무용품	19466000
지류	28039000
화방용품	72386500
NULL	119891500

```
SELECT 제품분류,SUM(매출합계)
FROM 문방구
GROUP BY ROLLUP(제품분류);
```

GROUPING SETS

GROUPING SETS는 각 인수들의 GROUP BY 결과를 합친 결과이다.

명령문 예

```
►SELECT JOB,DEPTNO,SUM(SAL)
FROM EMP
GROUP BY GROUPING SETS(JOB,DEPTNO);
```

JOB	DEPTNO	SUM(SAL)
CLERK	NULL	4150
SALESMAN	NULL	5600
PRESIDENT	NULL	5000
MANAGER	NULL	8275
ANALYST	NULL	6000
NULL	30	9400
NULL	20	10875
NULL	10	8750

직무	SUM(SAL)
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

GROUP BY JOB

DEPTNO	SUM(SAL)
10	8750
20	10875
30	9400

GROUP BY DEPTNO

GROUPING SETS(()) 혹은 GROUPING SETS (NULL) 은 전체에 대한 총합 행을 출력시킨다는 사실을 반드시 기억하자.

GROUPING SETS(A,B,C,D) 의 결과값은 GROUPING SETS(A) + GROUPING SETS(B) + GROUPING SETS(C) + GROUPING SETS(D) 와 동일한 결과를 출력한다.

ROLLUP : 인수가 두 개 이상인 경우

ROLLUP(A,B) 로 인수가 두개인 경우에는 ROLLUP의 결과값을 다음과 같이 구할 수 있다.

ROLLUP(A,B) = GROUPING SETS((A,B))+ GROUPINGS SETS(A) + GROUPING SETS(NULL)

정리 : 그룹함수 인수 2개인 경우

GROUP BY 그룹함수 (A,B)

ROLLUP : GROUP BY A,B + GROUP BY A + GROUP BY NULL

CUBE : GROUP BY A,B + GROUP BY A + GROUP BY B + GROUP BY NULL

GROUPING SETS : GROUP BY A + GROUP BY B

그룹함수의 특징

ROLLUP : ROLLUP의 인수는 계층구조이므로 인수 순서가 바뀌면 수행 결과도 바뀐다.

ROLLUP(DEPTNO,MGR) ≠ ROLLUP(MGR,DEPTNO)

CUBE : 다차원 집계 함수이므로, 시스템 부하가 ROLLUP 보다 크다

CUBE의 인수들은 평등한 관계이므로, 순서는 무관하다. (CUBE(DEPTNO,MGR) = CUBE(MGR, DEPTNO))

GROUPING SETS : GROUPING SETS의 인수들은 평등한 관계이므로, 순서는 무관하다. (GROUPING SETS(DEPTNO,MGR) = GROUPING SETS(MGR, DEPTNO))

윈도우 함수

개별 데이터들에 대한 연산 결과를 출력해주는 분석 함수(Analytical Function)

PARTITION BY : GROUP BY 와 비슷한 역할, 기준컬럼에 따라서 윈도우 함수 처리 범위를 모으는 역할

ORDER BY : 정렬작업으로 행과 행간의 관계를 정의해주는 역할

WINDOWING 절 : 값의 처리 범위를 정의

순위함수

함수	결과
RANK	중복하는 등수 존재, 중복이 있는 경우 연속하는 등수 없음
DENSE_RANK	중복하는 등수 존재, 중복이 있어도 다음 연속하는 등수
ROW_NUMBER	중복하는 등수 없다. 연속하는 등수

집계함수

► ***<집계함수>*** **(*<집계 대상 컬럼명>*) OVER (PARTITION BY *<집계할 대상의 범위>* ORDER BY *<정렬 기준 컬럼 >***
ROWS *<행의 수>* PRECEDING*)*

► ***<집계함수>*** **(*<집계 대상 컬럼명>*) OVER (PARTITION BY *<집계할 대상의 범위>* ORDER BY *<정렬 기준 컬럼 >***
ROWS BETWEEN *<행의 수1>* AND *<행의 수2>)*

► ***<집계함수>*** **(*<집계 대상 컬럼명>*) OVER (PARTITION BY *<집계할 대상의 범위>* ORDER BY *<정렬 기준 컬럼 >***
RANGE *<값>* PRECEDING*)*

► ***<집계함수>*** **(*<집계 대상 컬럼명>*) OVER (PARTITION BY *<집계할 대상의 범위>* ORDER BY *<정렬 기준 컬럼 >***
RANGE BETWEEN *<값1>* AND *<값2>*)

ENAME	SAL	DEPTNO	WIN_SUM
MILLER	1300	10	6300
KING	5000	10	8750
CLARK	2450	10	11750
FORD	3000	20	11550
ADAMS	1100	20	9550
SCOTT	3000	20	
JONES	2975	20	
SMITH	800	20	
JAMES	950	30	
TURNER	1500	30	
BLAKE	2850	30	
MARTIN	1250	30	
WARD	1250	30	
ALLEN	1600	30	

UNBOUNDED PRECEDING
PRECEDING : 이전의 행
CURRENT ROW 현재 행일 경우
FOLLOWING : 다음 나오는 행
UNBOUNDED FOLLOWING

행 순서 함수

FIRST_VALUE (<값>) : 윈도우에서 가장 처음에 나오는 값을 구한다.

LAST_VALUE (<값>) : 윈도우에서 가장 나중에 나오는 값을 구한다.
LAG (<값>, <숫자1>, <숫자2>) : 파티션에서 <값>에 대하여 현재행 이전 <숫자1>행의 값을 구한다. 그 값이 없다면 <숫자2>를 반환한다.
LEAD (<값>, <숫자1>, <숫자2>) :파티션에서 <값> 에 대하여 현재행 이후 <숫자1>행의 값을 구한다. 그 값이 없다면 <숫자2>를 반환한다.

비율 관련 함수

RATIO_TO_REPORT : 파티션 내 전체 SUM(컬럼)값에 대한 행별 칼럼 값의 백분율을 소수점으로 구할 수 있다.
CUME_DIST : 파티션 전체 건수에서 동일한 할당을 나누었을때, 누적 백분율을 조회한다. (누적 분포상 위치를 0~1사이 값을 가진다.)
PERCENT_RANK : 파티션에 제일 먼저 나온 것을 0으로 제일 늦게 나온 것을 1로 하여, 값이 아닌 행의 순서별 위치 백분율을 조회한다. <중위값, Q1,Q2,Q3>
NTILE(n) : 파티션 별로 N등분하여 조를 나눈다. 이때 나머지는 상위 조에게 순서대로 배정되게 한다.

조인(JOIN) : 수평결합

CROSS JOIN

조건 없이 2개의 테이블을 하나로 조인하는 것이다.
카테시안 곱(Cartesian Product)이 발생한다.
.

```
SELECT   A.EMPNO , A.ENAME , B.DEPTNO   ,
B.DNAME
FROM   EMP   A   , DEPT B
WHERE   B.DEPTNO = A.DEPTNO ;
```

```
SELECT   A.EMPNO , A.ENAME , B.DEPTNO   ,
B.DNAME
FROM   EMP   A   JOIN   DEPT B
ON     B.DEPTNO = A.DEPTNO ;
```

NATURAL JOIN의 특징

1. USING, ON, WHERE 절로 조인 조건 절을 서술할 필요가 없다.
2. 공통 컬럼을 자동적으로 찾아서 조인한다.
3. OUTER JOIN 에서도 사용 가능하다.

외부조인 (OUTER JOIN)

OUTER JOIN은 선행 테이블(좌측 테이블) 혹은 후행 테이블(우측 테이블)을 기준으로 조인 조건에 만족하지 않는 행들을 포함하여 출력한다.

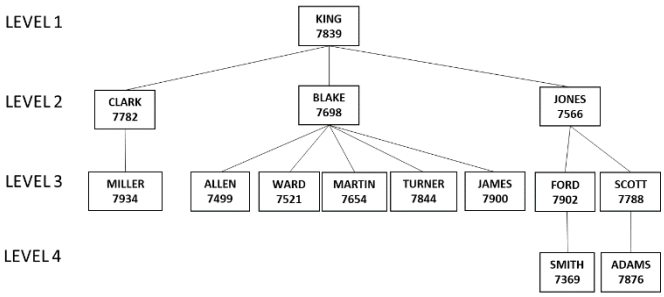
LEFT JOIN

▶SELECT
FROM ***<테이블 명 1>*** **LEFT OUTER JOIN** ***<테이블 명 2>*** **ON** ***<조인 조건식>***
▶SELECT
FROM ***<테이블 명 1>*** **LEFT OUTER JOIN** ***<테이블 명 2>*** **ON** ***<조인 조건식>***

RIGHT JOIN

▶SELECT
FROM ***<테이블 명 1>*** **RIGHT OUTER JOIN** ***<테이블 명 2>*** **ON** ***<조인 조건식>***
▶SELECT
FROM ***<테이블 명 1>,<테이블 명 2>***
WHERE ***<테이블 명1 컬럼>*** **(+) =** ***<테이블명2 컬럼>***

셀프조인(SELF JOIN)



부모노드에서 부터 자식노드까지 데이터들을 전개하는 것을 순방향 전개라고 한다.

셀프 조인에서는 두개 이상의 동일한 테이블을 이용해서 조인이 이루어지는데, 동일 테이블 각각의 역할이 다르다

계층형 질의

부모와 자식, 상사와 부하와 같은 구조를 계층형 구조라고 부른다. ORACLE에서는 이러한 계층형 데이터를 분석하기 위한 기능을 제공한다. 이를 계층형 질의라고 한다.
※ 계층형 질의는 SQL SERVER에서 지원하지 않는다

► SELECT EMPNO, ENAME, MGR,
CONNECT_BY_ISLEAF, LEVEL
FROM EMP
WHERE ***<조건절>***
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO=MGR;

START WITH : 계층 구조 전개의 시작위치 (루트 데이터, 부모 데이터의 정보)
CONNECT BY : 다음에 전개될 자식데이터를 지정한다. 계층형 전개의 조건식으로, 자식 데이터는 CONNECT BY 절에 주어진 조건을 만족해야 한다.
PRIOR : CONNECT BY 절에 사용되며, 이전 레벨 데이터를 지칭할 때 쓴다.
WHERE : 모든 전개 수행 후에, 지정된 조건을 만족하는 데이터만을 추출한다. ※ 계층 데이터 형성 후 가장 마지막에 적용되는 조건이다.

CONNECT BY PRIOR 자식컬럼 = 부모컬럼 ➔ 부모에서 자식으로 트리 그리기 : 순방향 전개

가상컬럼

LEVEL : 루트 데이터면 1 하위 데이터이면 2이다. 리프(Leaf) 데이터까지 1씩 증가한다.
CONNECT_BY_ISLEAF : 전개 과정에서 해당 데이터가 리프 데이터이면 1, 그렇지 않으면 0이다.
SYS_CONNECT_BY_PATH(<컬럼명>, '<경로분리자>') : 루트 데이터부터 현재 전개할 데이터까지의 경로를 표시한다.
CONNECT_BY_ROOT(<컬럼명>) : 현재 전개할 데이터의 루트 데이터를 표시한다.
※ CONNECT_BY_ROOT 에서 최상위 관리자 본인의 이름도 등장한다.

서브쿼리

쿼리문(메인쿼리) 안에 들어가는 쿼리문(서브쿼리) 쿼리문 작성시 사용되는 값을 다른 쿼리에서 구해야 할 경우 사용한다.

반환 결과에 따른 서브쿼리 분류

서브 쿼리 종류	설명
스칼라 서브쿼리 Scalar Sub Query	서브 쿼리의 결과가 1행, 1열인 서브쿼리를 의미한다. 값이 단 1개가 나온다는 특징이 있다.
단일 행 서브 쿼리 Single Row Sub Query	서브 쿼리의 결과가 단 1건 이하인 서브 쿼리를 의미한다. 단순한 값을 의미하므로 단일 행 비교하는 비교 연산자 = , < , <= , >, >= , < > 등이 있다.
다중행 서브 쿼리 Multi Row Sub Query	서브 쿼리의 실행 결과가 여러 행인 서브 쿼리를 의미하며 다중 행 서브 쿼리는 IN , ALL , ANY, SOME , EXIST 의 비교 연산자를 사용하여야 한다.
다중 칼럼 서브 쿼리 Multi Column Sub Query	서브 쿼리의 결과로 여러 칼럼을 반환하는 경우이다. 메인쿼리의 조건절에서 여러 칼럼을 동시에 비교할 수 있는데 이 때 서브 쿼리와 메인 쿼리에서 비교하고자 하는 칼럼의 수와 칼럼의 위치는 동일해야 한다.

INLINE VIEW (인라인뷰)
FROM 구에 SELECT문을 사용하여 가상의 테이블을 만드는 효과가 있다.

상호연관 서브쿼리

```
SELECT A.ENAME, A.SAL, A.DEPTNO
FROM EMP A
WHERE A.SAL > (SELECT AVG(B.SAL)
                FROM EMP B
                WHERE A.DEPTNO = B.DEPTNO);
```

다중행 서브쿼리 연산자

IN : 서브쿼리의 결과 중 하나라도 일치하면 조건은 TRUE
ANY,SOME : 서브쿼리의 결과와 하나이상 일치하면 조건은 TRUE
ALL : 서브쿼리의 결과와 모두 일치해야 조건이 참이된다.
EXISTS : 데이터의 존재 여부를 확인하는 쿼리에 해당한다. (만족하는 값이 없으면 공집합을 반환)

집합연산자 (수직결합)

UNION : 합집합 <중복된 것 배제, 정렬의 의미를 포함>
UNION ALL : 합집합, 중복된 데이터를 모두 가져온다. <정렬 없음>
INTERSECT : 교집합 <중복된 것 배제>
MINUS : 차집합 <MS SQL에서는 EXCEPT>, <중복된 것 배제>

테이블명, 컬럼명 명명 규칙

- 테이블명과 칼럼명은 반드시 문자로 시작해야한다.
- A-Z,a-z,0-9,
- 특수문자는 _, $, #만 사용 가능

제약조건
목적 : 데이터의 무결성 유지

UNIQUE KEY(고유키) : 고유키 정의
NOT NULL : NULL 값 입력금지
PRIMARY KEY(기본키) : UNIQUE & NOT NULL
CHECK : 입력 값 범위 제한
FOREIGN KEY(외래키) : NULL 가능,

테이블 생성
CREATE TABLE 테이블명 (컬럼명 데이터유형 제약조건);

테이블 구조 변경
- 컬럼 추가 : ALTER TABLE 테이블명 ADD(컬럼명 데이터유형);
- 삭제 : ALTER TABLE 테이블명 DROP COLUMN 컬럼명;
- 수정 : ALTER TABLE 테이블명 MODIFY (컬럼명 데이터유형 제약조건);

제약조건 삭제 : DROP CONSTRAINT 제약조건명;
제약조건 추가 : ADD CONSTRAINT 제약조건명 조건
테이블명 변경 : RENAME 변경전 테이블명 TO 변경후 테이블명;
테이블 삭제 : DROP TABLE 테이블명;
테이블 데이터 삭제 : TRUNCATE TABLE 테이블명;
컬럼명 변경 : RENAME COLUMN 변경전 컬럼명 TO 변경 후 컬럼명

INSERT

INSERT INTO TABLE (<컬럼명1>,<컬럼명2>,...)
VALUES(<값1>,<값2>,...);
VALUES의 로우 정보값
i) 문자열일 경우, '' 를 붙여서 입력
ii) 숫자일 경우, '' 없이 입력

1. 데이터의 추가
INSERT INTO EXAMPLE (EMPNO,ENAME,JOB)
VALUES(1000,'김강민','인사');
2. 컬럼목록을 생략하는 경우 데이터의 추가
INSERT INTO EXAMPLE
VALUES(1001, '이성우' ,'영업');
3. 컬럼 목록에 모든 컬럼이 있지 않는 경우
INSERT INTO EXAMPLE (EMPNO,ENAME)
VALUES(1002,'김영희');

UPDATE

UPDATE <테이블명>
SET <컬럼1>=<값1>, ...
WHERE <조건문>;

DELETE

DELETE FROM <테이블명>
WHERE <조건절>;

TCL

COMMIT : 트랜잭션을 완료하고 디스크에 반영한다. COMMIT 후, 복구는 불가하다.
ROLLBACK : 트랜잭션을 취소한다. 마지막 COMMIT 지점으로 돌아간다.
SAVEPOINT : ROLLBACK시 돌아가는 저장포인트를 지정한다. (세이브포인트 동일이름 지정시 덮여쓰기 됨)

ORACLE COMMIT

- DML(INSERT, UPDATE, DELETE) 는 사용자가 반드시 커밋 해주어야 데이터베이스에 반영된다.
- DDL(CREATE, ALTER, DROP, TRUNCATE) 는 사용시 자동적으로 커밋이 이루어진다.

SQL SERVER COMMIT

- AUTO COMMIT OFF : DML, DDL 모두 커밋 요함
- AUTO COMMIT ON : DML, DDL 모두 자동 커밋

DELETE VS TRUNCATE

DELETE : ROLLBACK 으로 복구 가능
TRUNCATE : 시스템 활용의 측면에서 TRUNCATE 시스템 부하가 적음

트랜잭션의 특성

원자성 : 트랜잭션에서 정의된 연산들은 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아있어야 한다.
고립성 : 트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다.

지속성 : 트랜잭션이 성공적으로 수행되면 그 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장된다.

일관성 : 트랜잭션이 실행되기 전의 데이터베이스 내용이 잘못 되어있지 않다면 트랜잭션이 실행된 이후에도 일관성 있는 데이터베이스를 유지한다.

DCL

GRANT : 데이터베이스 사용자에게 권한을 부여하는 것이다. 데이터베이스의 사용을 위해 데이터의 연결,수정,입력,삭제,조회 등을 하게 해준다.
REVOKE : 데이터베이스 사용자에게 부여된 권한을 회수한다.

GRANT *<권한명>* **ON** *<객체명>* **TO** *<유저명>*
REVOKE *<권한명>* **ON** *<객체명>* **FROM** *<유저명>*

ROLE

사용자에게 허가할 수 있는 권한들의 집합

ROLE을 이용하면 권한 부여와 회수를 쉽게 할 수 있다.
ROLE은 CREATE ROLE 권한을 가진 USER에 의해 생성된다.
한 사용자가 여러 개의 ROLE을 가질 수 있고, 여러 사용자에게 동일한 ROLE을 부여할 수 있다.
GRANT 와 REVOKE로 사용자에게 ROLE을 부여하고 취소도 가능하다.
사용자는 ROLE에 ROLE을 부여하는 것도 가능하다.

뷰(VIEW)

뷰는 데이터베이스에서 제공하는 가상의 테이블을 의미한다.
뷰를 사용하면 복잡한 쿼리문을 대신할 수 있기 때문에 개발의 용이성을 가진다.
뷰는 뷰를 만들 때 사용한 쿼리문을 저장하는 것이며, 뷰를 조회할 때는 뷰를 만들 때 사용한 쿼리문이 동작한다.

이론상 뷰의 장점(빈출) <독.편.보>

독립성 : 테이블 구조가 변경되어도 뷰를 사용하는 응용프로그램은 변경하지 않아도 된다.
편리성 : 복잡한 질의를 뷰로 생성함으로서 관련 질의를 단순하게 작성할 수 있다.
보안성 :숨기고 싶은 정보가 있다면 뷰를 생성할 시에 해당 컬럼을 빼고 생성하여 사용자에게 정보를 감출 수 있다.