

Content

- ◆ SQL Basic
- ◆ Operators
- ◆ Functions
- ◆ Join
- ◆ Aggregation
- ◆ Subqueries
- ◆ DDL
- ◆ DML

1. SQL Review

SQL Basics

- ◆ Sample Tables (of scott/TIGER user)

```
CREATE TABLE DEPT
(DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
 DNAME VARCHAR2(14),
 LOC VARCHAR2(13));
```

```
CREATE TABLE EMP
(EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
 ENAME VARCHAR2(10),
 JOB VARCHAR2(9),
 MGR NUMBER(4),
 HIREDATE DATE,
 SAL NUMBER(7,2),
 COMM NUMBER(7,2),
 DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT);
```

SQL Basics

dept

schema

열 이름	데이터 유형	널 가능	기본값	기본 키
DEPTNO	NUMBER(2,0)	No	-	1
DNAME	VARCHAR2(14)	Yes	-	-
LOC	VARCHAR2(13)	Yes	-	-

참조

emp

열 이름	데이터 유형	널 가능	기본값	기본 키
EMPNO	NUMBER(4,0)	No	-	1
ENAME	VARCHAR2(10)	Yes	-	-
JOB	VARCHAR2(9)	Yes	-	-
MGR	NUMBER(4,0)	Yes	-	-
HIREDATE	DATE	Yes	-	-
SAL	NUMBER(7,2)	Yes	-	-
COMM	NUMBER(7,2)	Yes	-	-
DEPTNO	NUMBER(2,0)	Yes	-	-

foreign key

편집	DEPTNO	DNAME	LOC
수정	10	ACCOUNTING	NEW YORK
수정	20	RESEARCH	DALLAS
수정	30	SALES	CHICAGO
수정	40	OPERATIONS	BOSTON

편집	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
수정	7369	SMITH	CLERK	7902	80/12/17	800	-	20
수정	7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
수정	7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
수정	7566	JONES	MANAGER	7839	81/04/02	2975	-	20
수정	7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
수정	7698	BLAKE	MANAGER	7839	81/05/01	2850	-	30

SQL Basics

◆ 기본 형식

```
SELECT [DISTINCT] {*, column_name [[AS] alias], . . . }  
FROM table_name  
WHERE condition;
```

- 예

```
SELECT *          // emp의 각 행에 대해 모든 컬럼 값을 출력  
FROM emp;
```

```
SELECT empno, ename // 지정된 두 컬럼 값들만 출력  
FROM emp;
```

5

SQL Basics

◆ 산술 연산자

- +, -, *, /

- 예

→ 산술연산 가능

```
SELECT empno, sal*12, comm, sal*12+comm  
FROM emp;
```

산술연산 후 결과는 return

EMPNO	SAL*12	COMM	SAL*12+COMM
7369	9600	-	-
7499	19200	300	19500
7521	15000	500	15500
7566	35700	-	-
7654	15000	1400	16400

- 주의: 컬럼 값이 NULL인 경우 산술 연산의 결과는 항상 NULL이 됨
 - ✓ NULL 값은 미확정이거나 알 수 없는 값을 의미
 - ✓ 산술연산이나 비교연산이 불가능함

6

SQL Basics

◆ Rename 연산자

- 컬럼 이름에 대한 별명(alias)을 부여
- 결과 출력 시 컬럼에 대한 heading으로 사용됨

- 예

```
SELECT ename, sal*12 AS Annual_Salary // AS 키워드는 생략 가능  
FROM emp;
```

```
SELECT ename, sal*12 "Annual Salary" // 공백이나 특수문자 포함 시  
FROM emp;
```

```
SELECT ename, sal*12+comm "연봉" // 한글 사용 시  
FROM emp;
```

7

SQL Basics

◆ Concatenation 연산자 (||)

- 문자열 타입의 컬럼 값을 하나의 문자열로 연결하여 결과 생성

- 예

```
SELECT ename, ' is a ', job // 세 개의 컬럼 출력  
FROM emp; // 'is a'도 하나의 컬럼으로 인식됨
```

ENAME	'ISA'	JOB
SMITH	is a	CLERK
ALLEN	is a	SALESMAN
WARD	is a	SALESMAN

```
SELECT ename || ' is a ' || job // 세 개의 컬럼이 하나의 컬럼으로  
FROM emp; // 연결되어 출력됨
```

ENAME 'ISA' JOB
SMITH is a CLERK
ALLEN is a SALESMAN
WARD is a SALESMAN

8

SQL Basics

◆ Distinct 키워드

- SELECT 문에 지정된 컬럼 리스트(tuple) 값들의 중복을 제거하여 출력

- 예

```
SELECT deptno  
FROM emp;
```

DEPTNO
20
30
30
20
30
30
10

```
SELECT DISTINCT deptno  
FROM emp;
```

DEPTNO
30
20
10

9

Operators

- 문자열 데이터 조회

```
SELECT *  
FROM emp  
WHERE ename = 'FORD'; // 문자열 값은 작은 따옴표 사용  
// 주의: 문자열 값은 대소문자를 구별하고 정확히 일치해야 함
```

광범위 인식방

- 날짜 데이터 조회

```
SELECT ename, hiredate  
FROM emp  
WHERE hiredate >= '1982/01/01'; // 날짜 값은 작은 따옴표 사용  
// 날짜는 내부적으로 숫자로 인식되므로 비교 연산자 사용 가능
```

Operators

◆ 조건 절 (WHERE 절)

- 검색 조건을 정의하여 조건을 만족하는 행들만 선택
- 비교, 논리, set membership (IN) 등의 연산자를 사용하여 조건을 표현

◆ 비교 연산자

- =, >, >=, <, <=, <>, !=
- 예

```
SELECT *  
FROM emp  
WHERE deptno = 10; // deptno 값이 10인 행들만 선택
```

```
SELECT ename, sal  
FROM emp  
WHERE sal >= 2000; // sal 값이 2000 이상인 행들만 선택
```

10

null → 무조건 거짓

Operators

◆ LIKE 연산자: 패턴을 이용한 문자열 검색

- wildcard 사용
 - %: 길이가 0 이상인 임의의 문자열에 대응
 - _: 임의의 한 문자에 대응

- 예

```
SELECT empno, ename  
FROM emp  
WHERE ename LIKE 'J%'; // J로 시작하는 문자열
```

EMPNO	ENAME
7566	JONES
7900	JAMES

```
SELECT empno, ename  
FROM emp  
WHERE ename LIKE '%A%'; // A를 포함하는 문자열
```

11

12

Operators

- 예 (계속)

```
SELECT empno, ename  
FROM emp  
WHERE ename LIKE '_A%';      // 두 번째 문자가 A인 문자열
```

```
SELECT empno, ename  
FROM emp  
WHERE ename LIKE '%\_%' ESCAPE '\';
```

- \를 escape character로 사용하여 \ 뒤에 오는 하나의 문자를 순수한 문자로 해석함
- 즉, 위 질의는 _를 포함하는 문자열을 검색

13

Operators

◆ 논리 연산자

- AND, OR, NOT

- 여러 개의 조건을 논리적으로 결합

- 예

```
SELECT empno, ename, job, deptno  
FROM emp
```

```
WHERE deptno=10 AND job='CLERK';
```

AND > OR (우선순위)

변경하고 싶으면
괄호 사용!

```
SELECT ename, hiredate  
FROM emp
```

```
WHERE hiredate >= '1982/01/01' OR job='CLERK';
```

```
SELECT empno, ename, job, deptno  
FROM emp  
WHERE NOT (deptno=10 AND job='CLERK');
```

14

Operators

◆ BETWEEN 연산자

- 검색하고자 하는 값의 최대 값과 최소 값을 설정

- 주의: 좌 우 경계 값도 포함됨

- 예

```
SELECT empno, ename, sal  
FROM emp  
WHERE sal BETWEEN 3000 AND 4000;
```



```
SELECT empno, ename, sal  
FROM emp  
WHERE sal >= 3000 AND sal <= 4000;
```

15

Operators

◆ Set Membership 연산자 (IN)

- 특정 컬럼의 값이 주어진 값 집합에 속하는지 여부를 검사

- 예

```
SELECT empno, ename, comm  
FROM emp  
WHERE comm IN (300, 500, 1400);  
// comm의 값이 300, 500, 1400 중의 하나인 행들만 선택
```

```
SELECT empno, ename  
FROM emp  
WHERE empno IN (SELECT DISTINCT mgr FROM emp);  
// 값 집합을 생성하기 위해 sub-query 사용 가능  
// 다른 사람의 manager 역할을 하고 있는 사람들만 선택
```

16

Operators

◆ NULL 값 검사 (IS NULL)

- 특정 컬럼의 값이 NULL 값을 갖는지를 검사
- 주의: = 연산자를 사용하면 안 됨

- 예

```
SELECT empno, mgr
FROM emp
WHERE mgr IS NULL;
// mgr의 값이 NULL인 행만 선택
// 주의: mgr = NULL로 표현하면 결과가 항상 거짓이 됨
```

17

Operators

◆ NOT 키워드 사용

- LIKE, BETWEEN, IN, IS NULL 연산자에 대해 NOT 연산자 적용 가능

- 예

```
SELECT empno, ename, comm
FROM emp
WHERE ename NOT LIKE '%A%'
AND sal NOT BETWEEN 500 AND 4000
AND comm NOT IN (300, 500, 1400)
AND mgr IS NOT NULL;
```

18

결과 정렬

◆ ORDER BY 절

- WHERE 절에 의해 선택된 행들을 주어진 기준(컬럼 값)에 의해 정렬 수행
- 오름차순 또는 내림차순으로 정렬 (default: 오름차순 정렬)
- 여러 개의 컬럼 값을 기준으로 정렬 가능
- 예

```
SELECT ename, sal
FROM emp
WHERE sal >= 2000
ORDER BY sal;          // sal 값에 따라 오름차순으로 정렬
```

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

19

결과 정렬

- 예

```
SELECT ename, sal
FROM emp
ORDER BY ename DESC;
→ ename 값에 대해 내림차순(알파벳의 역순)으로 정렬
```

```
SELECT empno, ename, sal
FROM emp
ORDER BY sal DESC, ename ASC;
```

→ sal의 내림차순으로 정렬하되, 같은 sal 값을 갖는 행들은 ename의 오름차순(알파벳 순)으로 정렬

20

참고: DUAL 테이블

DUAL

- SYS 사용자가 소유하는 테이블로, 모든 사용자들이 접근 가능
- DUMMY라는 이름을 갖는 문자열 컬럼에 하나의 문자가 저장되어 있음
- 특정 테이블에 관계없는 연산이나 함수의 실행 결과를 한 번만 출력하고자 할 때 사용

```
SELECT empno, sal*12  
FROM emp;
```

→ emp 테이블의 각 행마다 결과 출력(행의 개수만큼 반복 출력)

```
SELECT 10*20  
FROM emp;
```

```
SELECT 10*20  
FROM DUAL;  
→ DUAL 테이블에는 하나의 행만 있으므로 결과가 한 번만 출력됨
```

```
SELECT SYSDATE // 현재 시각  
FROM DUAL;
```

21

Functions – 숫자 함수

ROUND(반올림) 함수

```
SELECT ROUND(45.293, 2) // 소수점 이하 셋째 자리에서 반올림(45.29)  
FROM DUAL;
```

```
SELECT ROUND(45.293, -1) // 1의 자리에서 반올림 (50)  
FROM DUAL;
```

```
SELECT deptno, ROUND(AVG(sal), 2)  
FROM emp  
GROUP BY deptno;
```

DEPTNO	ROUND(AVG(SAL),2)
30	1566.67
20	2258.33
10	2916.67

TRUNC(버림) 함수

```
SELECT TRUNC(45.196, 2) // 소수점 이하 셋째 자리에서 버림 (45.19)  
FROM DUAL;
```

22

Functions – 숫자 함수

MOD(나머지) 함수

```
SELECT MOD(2850, 100) // 50  
FROM DUAL;
```

```
SELECT empno, ename // 사원번호가 홀수인 사원 정보  
FROM emp  
WHERE MOD(empno, 2) = 1;
```

EMPNO	ENAME
7369	SMITH
7499	ALLEN
7521	WARD
7839	KING

- 참고: CEIL(정수로 올림), FLOOR(정수로 내림) 함수

23

Functions – 문자 함수

UPPER 함수: 모든 문자를 대문자로 변환

```
SELECT UPPER('Welcome to Oracle 10g')  
FROM DUAL;  
→ WELCOME TO ORACLE 10G
```

```
SELECT UPPER(ename)  
FROM emp;
```

UPPER(ENAME)
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
KING
TURNER
JAMES

10개보다 많은 행을 사용할 수 있습니다. 행 선택기를 늘려 행을 더 표시하십시오.

24

Functions – 문자 함수

- ◆ LOWER 함수: 모든 문자를 소문자로 변환

```
SELECT LOWER('Welcome to Oracle 10g')
```

```
FROM DUAL;
```

→ welcome to oracle 10g

```
SELECT LOWER(ename)  
FROM emp;
```

LOWER(ENAME)
smith
allen
ward
jones
martin
blake
clark
king
turner
james

10개보다 많은 행을 사용할 수 있습니다. 행 선택기를 늘려 행을 더 표시하십시오.

25

Functions – 문자 함수

- ◆ LENGTH 함수: 문자열의 길이를 계산

```
SELECT LENGTH('Welcome to Oracle 10g')
```

```
FROM DUAL;
```

→ 21

```
SELECT ename, LENGTH(ename)  
FROM emp;
```

ENAME	LENGTH(ENAME)
SMITH	5
ALLEN	5
WARD	4
JONES	5
MARTIN	6
BLAKE	5
CLARK	5
KING	4
TURNER	6
JAMES	5

10개보다 많은 행을 사용할 수 있습니다. 행 선택기를 늘려 행을 더 표시하십시오.

26

Functions – 문자 함수

- ◆ INSTR 함수: 특정문자가 출현하는 위치를 찾음

```
SELECT INSTR('Welcome to Oracle 10g', 'o') // 첫 번째 출현 위치
```

```
FROM DUAL;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
W	e	1	c	o	m	e	t	o	O	r	a	c	1	e	1	0	g			→

```
SELECT INSTR('Welcome to Oracle 10g', 'o', 3, 2)
```

```
FROM DUAL; // 3번째 문자부터 검색하여 2번째 출현 위치
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
W	e	1	c	o	m	e	t	o	O	r	a	c	1	e	1	0	g			→

27

Functions – 문자 함수

- ◆ SUBSTR 함수: 문자의 일부분을 추출

```
SELECT SUBSTR('Welcome to Oracle 10g', 4, 3)
```

```
FROM DUAL; // 앞에서 4번째 문자부터 시작하여 길이가 3인 substring
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
W	e	1	c	o	m	e	t	o	O	r	a	c	1	e	1	0	g			→

```
SELECT hiredate, SUBSTR(hiredate, 1, 2) "입사연도"
```

```
FROM emp;
```

음수일 경우 뒤에서부터
역으로 계산

HIREDATE	입사연도
80/12/17	80
81/02/20	81
81/02/22	81
81/04/02	81
81/09/28	81
81/05/01	81

28

Functions – 문자 함수

- ◆ **LPAD** 함수: 지정된 크기의 공간 내에 오른쪽 맞춤 후 왼쪽 공백을 지정된 문자로 채움

```
SELECT LPAD('Oracle 10g', 20, '#')  
FROM DUAL;  
→ #####Oracle 10g
```

- ◆ **RPAD** 함수: 지정된 크기의 공간 내에 왼쪽 맞춤 후 오른쪽 공백을 지정된 문자로 채움

```
SELECT RPAD('Oracle 10g', 20, '#')  
FROM DUAL;  
→ Oracle 10g#####
```

Functions – 문자 함수

- ◆ **LTRIM** 함수: 왼쪽에서부터 지정된 문자를 삭제

- 주의: 문자가 지정되지 않을 경우 공백문자 삭제

```
SELECT LTRIM('aaaOracle 10gaaa', 'a') → 'Oracle 10gaaa'  
FROM DUAL;
```

```
SELECT LTRIM(' Oracle 10g ') → 'Oracle 10g '  
FROM DUAL;
```

- ◆ **RTRIM** 함수: 오른쪽에서부터 지정된 문자를 삭제

```
SELECT RTRIM('aaaOracle 10gaaa', 'a') → 'aaaOracle 10g'  
FROM DUAL;
```

Functions – 문자 함수

- ◆ **TRIM** 함수: **왼쪽 또는 오른쪽에서 문자 삭제**

- **LEADING, TRAILING, BOTH** : 문자 삭제 위치 지정(**생략 시 BOTH**)
- 삭제할 문자가 지정되지 않을 경우 **공백문자 삭제**

```
SELECT TRIM(LEADING 'a' FROM ename) // 왼쪽에서 'a' 문자들을 삭제  
FROM emp;
```

```
SELECT TRIM('a' FROM ename) // 양쪽에서 'a' 문자들을 삭제  
FROM emp;
```

```
SELECT TRIM(TRAILING FROM ename) // 오른쪽에서 공백 문자들을 삭제  
FROM emp;
```

```
SELECT TRIM(ename) // 양쪽에서 공백 문자들을 삭제  
FROM emp;
```

29

30

Functions – 문자 함수

- ◆ 조건절에서 문자 함수 사용

- 이름이 Smith인 직원의 정보를 검색
 - 저장된 데이터의 대소문자 사용 및 앞뒤 공백 존재 여부를 모를 때

```
SELECT * FROM emp  
WHERE TRIM(LOWER(ename)) = 'smith';  
// 또는 TRIM(UPPER(ename)) = 'SMITH';
```

- 이름의 마지막 글자가 n으로 끝나는 직원의 정보를 검색

```
SELECT * FROM emp  
WHERE SUBSTR(ename, -1, 1) = 'n';  
// 또는 ename LIKE '%n';
```

31

32

Functions – 날짜 함수

- ◆ SYSDATE 함수: 시스템의 현재 날짜 및 시각을 반환 (Date type)

```
SELECT SYSDATE  
FROM DUAL;
```

SYSDATE
08/25/2014

```
SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "NOW"  
FROM DUAL;
```

NOW
08-25-2014 20:20:10

minute

형식을 지정해 CHAR 타입으로 변경

33

Functions – 날짜 함수

- ◆ SYSDATE 함수

– 날짜 연산

```
SELECT SYSDATE - 1, SYSDATE + 1 // -24시간, +24시간  
FROM DUAL;
```

```
SELECT ename, hiredate, ROUND(SYSDATE - hiredate, 2)  
FROM emp; // 두 시각의 차이 계산(단위: 일)
```

ENAME	HIREDATE	ROUND(SYSDATE-HIREDATE,2)
SMITH	12/17/1980	12304.87
ALLEN	02/20/1981	12239.87
WARD	02/22/1981	12237.87
JONES	04/02/1981	12198.87
MARTIN	09/28/1981	12019.87
BLAKE	05/01/1981	12169.87
CLARK	06/09/1981	12130.87
KING	11/17/1981	11960.87

34

Functions – 날짜 함수

- ◆ MONTHS_BETWEEN 함수: 두 날짜 사이의 개월 수 계산

```
SELECT MONTHS_BETWEEN(TO_DATE('09-01-2014','MM-DD-YYYY'),  
                      TO_DATE('06-15-2014','MM-DD-YYYY')) "Months"  
FROM DUAL;
```

Months
2.5483870967741935483870967741935483871

```
SELECT ename, hiredate, ROUND(MONTHS_BETWEEN(SYSDATE, hiredate), 2)  
FROM emp;
```

ENAME	HIREDATE	ROUND(MONTHS_BETWEEN(SYSDATE,HIREDATE),2)
SMITH	12/17/1980	404.29
ALLEN	02/20/1981	402.19
WARD	02/22/1981	402.12
JONES	04/02/1981	400.77
MARTIN	09/28/1981	394.93
BLAKE	05/01/1981	399.8

35

Functions – 날짜 함수

- ◆ ADD_MONTHS 함수: 지정된 개월 수를 더함

```
SELECT hiredate, ADD_MONTHS(hiredate, 6)  
FROM emp  
WHERE ename='SMITH';
```

HIREDATE	ADD_MONTHS(HIREDATE,6)
12/17/1980	06/17/1981

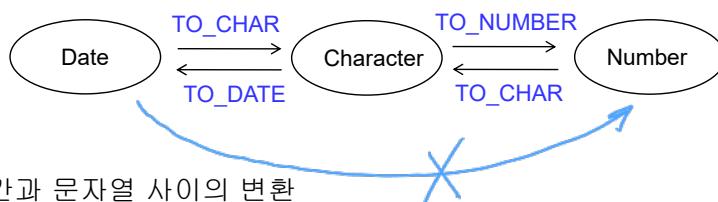
- ◆ NEXT_DAY 함수: 지정된 날짜 이후의, 지정된 요일에 해당되는 첫 날짜 계산

```
SELECT TO_CHAR(SYSDATE, 'YY/MM/DD DAY') "Now",  
      TO_CHAR(NEXT_DAY(SYSDATE, 'Fri'), 'YY/MM/DD DAY') "Next Friday"  
FROM DUAL;
```

Now	Next Friday
14/08/25 MONDAY	14/08/29 FRIDAY

36

Functions – 형 변환 함수



- ◆ 시간과 문자열 사이의 변환
 - 시간 형식(Datetime Format) 사용

종류	의미
YYYY	년도(4자리)
YY	년도(2자리)
MM	월(01~12)
MON	월을 알파벳 축약형으로 표현
DAY	요일
DY	요일을 축약형으로 표현

종류	의미
AM 또는 PM	오전/오후 표시
HH12 또는 HH	시간(01~12)
HH24	시간(00~23)
MI	분(00~59)
SS	초(00~59)
FM	앞의 0을 제거

37

Functions – 형 변환 함수

◆ 예

```
SELECT SYSDATE - TO_DATE('2014/01/01', 'YYYY/MM/DD')
```

FROM DUAL;

생략 가능

SYSDATE-TO_DATE('2014/01/01','YYYY/MM/DD')
236.852523148148148148148148148148148148

```
SELECT TO_CHAR(SYSDATE +3/24, 'YYYY"년" MM"월" DD"일" HH24"시" MI"분"')
```

FROM DUAL; // 현재 시각에 3시간을 더함

TO_CHAR(SYSDATE+3/24,'YYYY"년"MM"월"DD"일"HH24"시"MI"분"')
2014년 08월 25일 23시 28분

38

Functions – 형 변환 함수

- ◆ 숫자와 문자열 사이의 변환
 - 숫자 형식(Number Format) 사용

종류	의미	예	결과
9	한자리 숫자 표현	(1111, '99999')	1111
0	앞부분 공백을 0으로 표현	(1111, '099999')	001111
\$ 또는 L	달러 또는 현지통화기호	(1111, '\$99999')	\$1111
.	소수점의 위치	(1111, '99999.99')	1111.00
,	특정 위치에 , 표시	(1111, '99,999')	1,111
MI	음수에 대해 오른쪽이 – 표시	(-1111, '99999MI')	1111-
PR	음수에 대해 <>로 표시	(-1111, '99999PR')	<1111>
EEEE	과학적 표기법으로 표시	(1111, '9.999EEEE')	1.111E+03
V	10 ⁿ 을 곱한 값으로 표현	(1111, '9999V99')	111100
B	공백을 0으로 표현	(1111, 'B9999.99')	1111.00

39

Functions – 형 변환 함수

– 예

```
SELECT ename, TO_CHAR(sal, 'L999,999')
```

FROM emp;

ENAME	TO_CHAR(SAL,'L999,999')
SMITH	₩800
ALLEN	₩1,600
WARD	₩1,250
JONES	₩2,975
MARTIN	₩1,250
BLAKE	₩3,850

```
SELECT * FROM emp  
WHERE sal > TO_NUMBER('1,000.55', '9,999.99');
```

↑
형식을 갖는 숫자 값이 인수로 주어질 경우

40

Functions – NULL 변환

◆ NVL: NULL값을 지정된 값으로 변환

```
SELECT ename, sal, comm, sal*12 + NVL(comm, 0)
FROM emp;
```

ENAME	SAL	COMM	SAL*12+NVL(COMM,0)
SMITH	800	-	9600
ALLEN	1600	300	19500
WARD	1250	500	15500
JONES	2975	-	35700
MARTIN	1250	1400	16400
BLAKE	2300		24000

Comm이 null이면,
0으로 대체하여 계산한다

SELECT ename, NVL(TO_CHAR(mgr, '9999'), 'CEO')
FROM emp; // mgr은 NUMBER 타입이므로 문자열로 변환 필요

CLARK	7839
SCOTT	7566
KING	CEO
TURNER	7698

문자열 타입

- 참고: NVL2(e1, e2, e3) 함수
n번의 경우
n번이 있을 경우

41

Functions – DECODE / CASE

◆ CASE

- DECODE와 유사하나 다양한 비교 연산자를 사용하여 조건 표현 가능
- 예

```
SELECT AVG(CASE WHEN sal > 5000 THEN 5000
WHEN sal > 2000 THEN sal
ELSE 2000
END) "Average Salary"
```

FROM emp;

→ sal 값이 5000보다 크면 5000으로, 2000보다 작으면 2000으로 간주하고 sal들의 평균값을 계산

Functions – DECODE / CASE

◆ DECODE

- 주어진 컬럼의 값에 따라 서로 다른 값을 반환
- 예

```
SELECT ename, deptno,
DECODE (deptno, 10, 'ACCOUNTING',
20, 'RESEARCH',
30, 'SALES',
40, 'OPERATIONS', 'UNKNOWN') AS dname
FROM emp;
```

ENAME	DEPTNO	DNAME
SMITH	20	RESEARCH
ALLEN	30	SALES
WARD	30	SALES
JONES	20	RESEARCH
MARTIN	30	SALES
BLAKE	30	SALES
CLARK	10	ACCOUNTING
KING	10	ACCOUNTING

42

Join

◆ 기본 개념

- 관계형 데이터베이스는 일반적으로 여러 개의 테이블들로 구성되며 테이블에 저장된 데이터 사이에 관계(relationship)가 존재함
 - 예: 사원은 부서에 속함
- 데이터 검색 시 여러 테이블에 저장된 데이터들을 대상으로 검색 수행 필요
 - 예: 사원들의 개인정보 뿐만 아니라 소속 부서에 대한 정보도 검색
- 하나의 SQL 질의에서 여러 테이블들을 결합하여 검색하기 위해 Join을 실행

emp 테이블

열 이름	데이터 유형
EMPNO	NUMBER(4,0)
ENAME	VARCHAR2(10)
JOB	VARCHAR2(9)
MGR	NUMBER(4,0)
HIREDATE	DATE
SAL	NUMBER(7,2)
COMM	NUMBER(7,2)
DEPTNO	NUMBER(2,0)

dept 테이블

열 이름	데이터 유형
DEPTNO	NUMBER(2,0)
DNAME	VARCHAR2(14)
LOC	VARCHAR2(13)

참조

Join

- Join 예

emp table *dept table*

번호	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	80/12/17	800	-	20
2	7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
3	7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
4	7566	JONES	MANAGER	7839	81/04/02	2975	-	20
5	7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
6	7698	BLAKE	MANAGER	7839	81/05/01	2850	-	30

번호	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

45

Join

◆ Cartesian Product(곱 집합)

- Join 조건을 명시하지 않을 경우 두 테이블의 행들을 가능한 모든 방법으로 결합함

```
SELECT *
```

```
FROM emp, dept;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7902	80/12/17	800	-	20	10	ACCOUNTING	NEW YORK
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30	10	ACCOUNTING	NEW YORK
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	7839	81/04/02	2975	-	20	10	ACCOUNTING	NEW YORK
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30	10	ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	7839	81/05/01	2850	-	30	10	ACCOUNTING	NEW YORK
7782	CLARK	MANAGER	7839	81/06/09	2450	-	10	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT	-	81/11/17	5000	-	10	10	ACCOUNTING	NEW YORK
7844	TIRMAN	SUPPORTMAN	7839	81/05/01	1500	0	30	10	ACCOUNTING	NEW YORK

- 일반적으로 join 조건을 사용하여 의미있는 행들끼리만 결합해야 함

46

Join

◆ Equi-Join

- 가장 일반적인 join 방법으로, 각 테이블에서 하나씩 컬럼을 선택하여 그 값들이 같은 행들만 서로 연결시킴
- 예

```
SELECT *
```

```
FROM emp, dept
```

```
WHERE emp.deptno = dept.deptno; // join을 위한 조건
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	7902	80/12/17	800	-	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30	30	SALES	CHICAGO
7566	JONES	MANAGER	7839	81/04/02	2975	-	20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	7839	81/05/01	2850	-	30	30	SALES	CHICAGO
7782	CLARK	MANAGER	7839	81/06/09	2450	-	10	10	ACCOUNTING	NEW YORK

47

Join

- 두 테이블에서 join하는 컬럼 이름이 같을 경우,

- 테이블 이름이나 별칭(tuple 번수)을 사용하여 구별
- SELECT 절에서 중복된 컬럼은 한 번만 사용

```
SELECT e.ename, d.dname ← tuple 번수 e, d 생략 가능
```

```
FROM emp e, dept d
```

(각 컬럼의 소속이 명확할 경우)

```
WHERE e.deptno = d.deptno;
```

또는

```
SELECT e.ename, d.dname
```

```
FROM emp e JOIN dept d
```

```
ON e.deptno = d.deptno;
```

ENAME	DNAME
SMITH	RESEARCH
ALLEN	SALES
WARD	SALES
JONES	RESEARCH
MARTIN	SALES

48

Join

- WHERE 절에는 join 조건 외의 다른 조건들도 올 수 있음

```
SELECT ename, dname
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND emp.ename = 'SCOTT';
```

49

Join

◆ Non-Equi-Join

- Join 조건으로 동등 비교 외의 다른 연산자를 사용 예:

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal >= s.losal AND e.sal <= s.hisal; // BETWEEN 사용 가능
```

실행 결과

salgrade 테이블

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1
WARD	1250	2
MARTIN	1250	2
MILLER	1300	2
TURNER	1500	3
ALLEN	1600	3
CLARK	2450	4

50

Join

- 세 개의 테이블 join 예
- ```
SELECT e.ename, d.dname, s.grade
FROM emp e, dept d, salgrade s
WHERE e.deptno = d.deptno // e와 d의 join 조건
 AND e.sal BETWEEN s.losal AND s.hisal; // e와 s의 join 조건
```

→ 각 사원에 대해 이름, 부서명, 급여 등급을 출력

| ENAME  | DNAME      | GRADE |
|--------|------------|-------|
| SMITH  | RESEARCH   | 1     |
| JAMES  | SALES      | 1     |
| WARD   | SALES      | 2     |
| MARTIN | SALES      | 2     |
| MILLER | ACCOUNTING | 2     |
| TURNER | SALES      | 3     |
| ALLEN  | SALES      | 3     |

51

# Join

## ◆ Self Join

- 하나의 테이블을 자기 자신과 join 수행
- 예: 각 사원에 대해 사원이름과 manager 이름을 함께 검색

```
SELECT e.ename || '의 매니저는' || m.ename
FROM emp e, emp m
WHERE e.mgr = m.empno;
```

emp e                                  emp m

| EMPNO | ENAME  | MGR  | EMPNO | ENAME  | MGR  |
|-------|--------|------|-------|--------|------|
| 7369  | SMITH  | 7902 | 7369  | SMITH  | 7902 |
| 7499  | ALLEN  | 7698 | 7499  | ALLEN  | 7698 |
| 7521  | WARD   | 7698 | 7521  | WARD   | 7698 |
| 7566  | JONES  | 7839 | 7566  | JONES  | 7839 |
| 7654  | MARTIN | 7698 | 7654  | MARTIN | 7698 |
| 7698  | BLAKE  | 7839 | 7698  | BLAKE  | 7839 |
| 7782  | CLARK  | 7839 | 7782  | CLARK  | 7839 |
| 7839  | KING   | -    | 7839  | KING   | -    |

| E.ENAME    '의 매니저는'    M.ENAME |
|--------------------------------|
| SMITH의 매니저는 FORD               |
| ALLEN의 매니저는 BLAKE              |
| WARD의 매니저는 BLAKE               |
| JONES의 매니저는 KING               |
| MARTIN의 매니저는 BLAKE             |
| BLAKE의 매니저는 KING               |
| CLARK의 매니저는 KING               |

52

null값이나 투표는 join 안됨

# Join

## ◆ Outer Join

- 회사의 사장은 mgr 값이 NULL이므로, 앞의 질의에서 join 시 배제되어 최종 결과에 나타나지 않음

- Outer Join은 join 조건을 만족하지 않는 행들도 결과에 포함시킴

- 예 1 (Left Outer Join)

```
SELECT e.ename || '의 매니저는' || m.ename
FROM emp e, emp m
WHERE e.mgr = m.empno (+);
```

→ e.mgr에 대응되는 m.empno가 없는 경우에도 왼쪽 테이블의 행을 결과에 포함시킴

다른 표현:

```
SELECT e.ename, m.ename
FROM emp e LEFT OUTER JOIN emp m
ON e.mgr = m.empno
```

| E.ENAME  '의 매니저는'  M.ENAME |
|----------------------------|
| SMITH의 매니저는 FORD           |
| ALLEN의 매니저는 BLAKE          |
| WARD의 매니저는 BLAKE           |
| JONES의 매니저는 KING           |
| MARTIN의 매니저는 BLAKE         |
| BLAKE의 매니저는 KING           |
| CLARK의 매니저는 KING           |
| KING의 매니저는                 |
| TURNER의 매니저는 BLAKE         |

53

# Join

## - 예 2 (Right Outer Join)

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno (+) = d.deptno;
```

→ emp 테이블에 나타나지 않는 부서 번호에 대해서도 dept 테이블의 dname을 결과에 포함시킴 (사원이 한 명도 없는 부서)

다른 표현:

```
SELECT e.ename, d.dname
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno = d.deptno
```

| ENAME  | DNAME      |
|--------|------------|
| KING   | ACCOUNTING |
| CLARK  | ACCOUNTING |
| MILLER | ACCOUNTING |
| FORD   | RESEARCH   |
| SMITH  | RESEARCH   |
| JONES  | RESEARCH   |
| JAMES  | SALES      |
| TURNER | SALES      |
| MARTIN | SALES      |
| WARD   | SALES      |
| ALLEN  | SALES      |
| BLAKE  | SALES      |
| -      | OPERATIONS |

54

# Aggregation

## ◆ 그룹 함수(aggregate function)

- 여러 행들을 그룹으로 묶은 후, 특정 열에 대해 계산을 수행한 결과를 반환

| 함수    | 설명                |
|-------|-------------------|
| COUNT | 행의 개수를 구함         |
| SUM   | 해당 열의 행들의 합을 구함   |
| AVG   | 해당 열의 행들의 평균값을 구함 |
| MIN   | 해당 열의 행들의 최소값을 구함 |
| MAX   | 해당 열의 행들의 최대값을 구함 |

- 주로 GROUP BY 절과 같이 사용됨

- GROUP BY 절이 없을 경우 모든 행들을 하나의 그룹으로 간주

- HAVING 절에서도 사용될 수 있음

- 주의: 함수의 계산 시 NULL 값을 제외하고 계산함

- 단, COUNT(\*)는 예외적으로 NULL 값 여부와 관계 없이 모든 행들의 개수를 계산

55

# Aggregation

## ◆ 예

```
SELECT COUNT(comm)
FROM emp; // comm이 NULL이 아닌 행들의 개수
// 전체 행의 개수와 일치하지 않을 수 있음
```

```
SELECT COUNT(job)
FROM emp; // job이 NULL이 아닌 행들의 개수
// 실제 job의 값은 중복 저장되어 있음
```

```
SELECT COUNT(DISTINCT job)
FROM emp; // 사원들에게 부여된 서로 다른 job의 수
```

```
SELECT COUNT(*)
FROM emp; // emp 내에 저장된 전체 행의 개수
// 모든 컬럼 값이 NULL인 행들도 포함
```

56

# Aggregation

WHERE 절에서는 쓸 수 있다!

```
SELECT SUM(sal), AVG(sal), MAX(sal), MIN(sal)
FROM emp; // 급여 값들의 합, 평균, 최대, 최소
```

```
SELECT ename, SUM(sal)
FROM emp;
```

→ 오류 발생!

SUM(sal)의 결과 값은 모든 행들에 대해 계산된 하나의 값이므로 특정 ename과 대응될 수 없음

# Aggregation

## ◆ GROUP BY 줄

- 특정 컬럼들의 값을 기준으로 전체 행들을 분류(grouping)함
  - GROUP BY 절에 포함되지 않은 컬럼은 SELECT 절에서 사용될 수 없음
  - 예:

```
SELECT deptno, AVG(sal)
FROM emp
```

## GROUP BY deptno

→ 각 부서번호마다 sal 값들의 평균을 계산하여 출력

# Aggregation

```
SELECT deptno, mgr, COUNT(sal), AVG(sal)
FROM emp
GROUP BY deptno, mgr;
```

→ 서로 다른 (deptno, mgr) 쌍마다 sal의 개수 및 평균을 계산

| DEPTH | MGR  | COUNT(SAL) | AVG(SAL) |
|-------|------|------------|----------|
| 20    | 7839 | 1          | 2975     |
| 10    | 7839 | 1          | 2450     |
| 30    | 7698 | 5          | 1310     |
| 20    | 7566 | 1          | 3000     |
| 10    | 7782 | 1          | 1300     |
| 20    | 7902 | 1          | 800      |
| 10    | -    | 1          | 5000     |
| 30    | 7839 | 1          | 2850     |

# Aggregation

#### ◆ HAVING 절

- GROUP BY 절에 의해 생성된 그룹들을 대상으로 특정 조건에 맞는 그룹들만 선택함

—

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
```

▶ 균여 평균이 2000 이상인 그룹들만 선택하여 결과 출력

# Aggregation

```

SELECT deptno, AVG(sal)
FROM emp
WHERE sal >= 1000
GROUP BY deptno
HAVING AVG(sal) >= 2000;

```

— ⑤  
— ①  
— ②  
— ③  
— ④



- ① 사원 테이블에서
- ② 1000 이상의 급여를 받는 사람들만 대상으로
- ③ 부서별로 그룹을 나눠
- ④ 평균을 구한 후, 부서별 평균 급여가 2000 이상인 부서들에 대해서만
- ⑤ 그 결과를 출력함

실습 #1 - 8번 문제 (④은 필요X)

61

# Aggregation

## ◆ LISTAGG 함수

예:

```

SELECT deptno "DeptNo",
 LISTAGG(ename, ',') WITHIN GROUP (ORDER BY ename DESC)
 "Emps"
FROM emp
GROUP BY deptno
ORDER BY deptno;

```

→ emp 행들을 deptno를 기준으로 그룹으로 나눈 후, 각 그룹의 행들을 ename 값의 역순으로 정렬하고 ename 값을 하나로 합쳐서 출력(구분자 이용)

| DeptNo | Emps                                      |
|--------|-------------------------------------------|
| 10     | MILLER, KING, CLARK                       |
| 20     | SMITH, SCOTT, JONES, FORD, ADAMS          |
| 30     | WARD, TURNER, MARTIN, JAMES, BLAKE, ALLEN |

63

# Aggregation

## ◆ LISTAGG 함수

- 테이블의 모든 행 또는 GROUP BY 절에 의해 생성된 각 그룹에 속한 행들을 정렬한 후 지정된 컬럼의 값을 하나의 값으로 concatenation 함

예:

문자열  

```

SELECT LISTAGG(ename, ',') 구분
 WITHIN GROUP (ORDER BY hiredate, ename) "Emp_list",
 MIN(hiredate) "Earliest"
FROM emp
WHERE deptno = 30;

```

→ emp 테이블의 모든 행들을 (hiredate, ename)으로 정렬하고  
그 순서대로 ename 값을 하나의 값으로 합쳐서 출력(구분자 이용)

| Emp_list                                  | Earliest |
|-------------------------------------------|----------|
| ALLEN; WARD; BLAKE; TURNER; MARTIN; JAMES | 81/02/20 |

62

# Example tables

## dept

| 열 이름   | 데이터 유형       | 널 가능 | 기본값 | 기본 키 |
|--------|--------------|------|-----|------|
| DEPTNO | NUMBER(2,0)  | No   | -   | 1    |
| DNAME  | VARCHAR2(14) | Yes  | -   | -    |
| LOC    | VARCHAR2(13) | Yes  | -   | -    |

참조

| 편집 | DEPTNO     | DNAME    | LOC |
|----|------------|----------|-----|
| 10 | ACCOUNTING | NEW YORK |     |
| 20 | RESEARCH   | DALLAS   |     |
| 30 | SALES      | CHICAGO  |     |
| 40 | OPERATIONS | BOSTON   |     |

## emp

| 열 이름     | 데이터 유형       | 널 가능 | 기본값 | 기본 키 |
|----------|--------------|------|-----|------|
| EMPNO    | NUMBER(4,0)  | No   | -   | 1    |
| ENAME    | VARCHAR2(10) | Yes  | -   | -    |
| JOB      | VARCHAR2(9)  | Yes  | -   | -    |
| MGR      | NUMBER(4,0)  | Yes  | -   | -    |
| HIREDATE | DATE         | Yes  | -   | -    |
| SAL      | NUMBER(7,2)  | Yes  | -   | -    |
| COMM     | NUMBER(7,2)  | Yes  | -   | -    |
| DEPTNO   | NUMBER(2,0)  | Yes  | -   | -    |

foreign key

| 편집   | EMPNO  | ENAME    | JOB  | MGR      | HIREDATE | SAL  | COMM | DEPTNO |
|------|--------|----------|------|----------|----------|------|------|--------|
| 7369 | SMITH  | CLERK    | 7902 | 80/12/17 | 800      | -    | 20   |        |
| 7499 | ALLEN  | SALESMAN | 7698 | 81/02/20 | 1600     | 300  | 30   |        |
| 7521 | WARD   | SALESMAN | 7698 | 81/02/22 | 1250     | 500  | 30   |        |
| 7566 | JONES  | MANAGER  | 7839 | 81/04/02 | 2975     | -    | 20   |        |
| 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250     | 1400 | 30   |        |
| 7698 | BLAKE  | MANAGER  | 7839 | 81/05/01 | 2850     | -    | 30   |        |

64

실습 #1 - 9번 문제

# Subqueries

## ◆ Nested subqueries

- 질의(main query) 안에 포함된 부질의
- 일반적으로 set membership/comparison/cardinality 등을 평가하기 위해 사용됨
- single row(단일 행) subqueries
  - 결과가 하나의 행
  - main query에서 비교 연산자(=, <, > 등)와 함께 사용됨
- multiple rows(다중 행) subqueries
  - 결과가 여러 개의 행들의 집합
  - main query에서 비교 연산자 및 In, All, Any/Some, Exist 등의 연산자와 함께 사용됨

65

# Subqueries

## ◆ Single row subqueries

- 예 1: JONES가 속한 부서의 이름은?

```
SELECT dname
FROM dept
WHERE deptno = (SELECT deptno
 FROM emp
 WHERE ename = 'JONES');
```

main query

subquery

- subquery의 결과가 하나의 행일 경우에만 실행됨

- ✓ 결과가 여러 행인 경우 오류 발생

```
SELECT dname
FROM dept
WHERE deptno = (SELECT deptno
 FROM emp
 WHERE ename LIKE 'J%');
```

➔ ORA-01427: single-row subquery returns more than one row

66

# Subqueries

## ◆ Single row subqueries

- 예 1: JONES가 속한 부서의 이름은?
  - 다음과 같은 join 질의로 표현 가능

```
SELECT DISTINCT dname
FROM dept JOIN emp
ON dept.deptno = emp.deptno
WHERE ename = 'JONES' ;
```

67

# Subqueries

## ◆ Single row subqueries

- 예 2: 전체 사원의 평균 급여보다 더 많은 급여를 받는 사원은?

```
SELECT ename, sal
FROM emp
WHERE sal > (SELECT AVG(sal)
 FROM emp);
```

- 위 subquery의 결과는 하나의 값이므로 항상 실행 가능
- join 질의로 표현 가능한가??

68

# Subqueries

## ◆ Multiple rows subqueries

- subquery의 결과가 여러 행들의 집합일 경우
- main query에서는 반드시 다음과 같은 다중 행 연산자와 함께 사용해야 함

| 연산자      | 의미                                           |
|----------|----------------------------------------------|
| IN       | 비교 연산이 = 이고, subquery의 결과 값들 중 어느 하나와 일치하면 참 |
| ANY/SOME | 비교 조건이 subquery의 결과 값들 중 어느 하나와 만족하면 참       |
| ALL      | 비교 조건이 subquery의 결과 값들 모두와 만족해야만 참           |
| EXISTS   | subquery의 결과가 하나 이상 존재하면 참                   |

69

# Subqueries

## ◆ IN 연산자

- 예: 급여를 3000 이상 받는 사원이 있는 부서에 근무하는 사원은?

```
SELECT ename, sal, deptno
FROM emp
```

```
WHERE deptno IN
```

```
(SELECT deptno
 FROM emp
 WHERE sal >= 3000);
```

→ 급여가 3000 이상인 사원이 속한 부서번호들의 집합

- 다음과 같은 self join 질의로 표현 가능

```
SELECT DISTINCT e.ename, e.sal, e.deptno
FROM emp e JOIN emp f ON e.deptno = f.deptno
WHERE f.sal >= 3000
```

- c.f. 급여를 3000 이상 받는 사원과 같은 부서에 근무하는 사원은?

70

# Subqueries

## ◆ ALL 연산자

- 예 1: 30번 부서에 속한 모든 사원들보다 더 많은 급여를 받는 사원은?

```
SELECT ename, sal
FROM emp
WHERE sal > ALL (SELECT sal
 FROM emp
 WHERE deptno = 30);
```

→ 30번 부서에 속한 모든 사원들의 급여 집합

- 다음과 같이 MAX 함수를 사용한 단일 행 subquery로 표현 가능

```
SELECT ename, sal
FROM emp
WHERE sal > (SELECT MAX(sal)
 FROM emp
 WHERE deptno = 30);
```

# Subqueries

## ◆ ALL 연산자

- 예 2: 같은 부서에 급여를 3000 이상 받는 사원이 없는 사원은?

(= 급여를 3000 이상 받는 사원이 있는 부서에 속하지 않는 사원)

```
SELECT ename
FROM emp
WHERE deptno NOT IN (SELECT deptno
 FROM emp
 WHERE sal >= 3000);
```

- 다음 질의와 동일

```
SELECT ename
FROM emp
WHERE deptno <> ALL (SELECT deptno
 FROM emp
 WHERE sal >= 3000);
```

71

# Subqueries

## ◆ ANY/SOME 연산자

- 예 1: 30번 부서에 속한 어떤 사람보다 더 많은 급여를 받는 사원은?

```
SELECT ename, sal
FROM emp
WHERE sal > ANY (SELECT sal
 FROM emp
 WHERE deptno = 30);
```

- 다음과 같이 MIN 함수를 사용한 단일 행 subquery로 표현 가능

```
SELECT ename, sal
FROM emp
WHERE sal > (SELECT MIN(sal)
 FROM emp
 WHERE deptno = 30);
```

73

# Subqueries

## ◆ ANY/SOME 연산자

- 예 2: 급여를 3000 이상 받는 사원과 같은 부서에 근무하는 사원은?

```
SELECT ename
FROM emp
WHERE deptno IN (SELECT deptno
 FROM emp
 WHERE sal >= 3000);
```

- 다음 질의와 동일

```
SELECT ename
FROM emp
WHERE deptno = ANY (SELECT deptno
 FROM emp
 WHERE sal >= 3000);
```

74

# Subqueries

## ◆ EXISTS 연산자

- 예: 직무가 "CLERK"인 사원이 있는 부서의 이름은?

1. 

```
SELECT dname
 FROM dept d // d: correlation variable
 WHERE EXISTS (SELECT *
 FROM emp
 WHERE deptno = d.deptno
 AND job = 'CLERK');
```

2. 

```
SELECT dname
 FROM dept
 WHERE deptno IN (SELECT deptno
 FROM emp
 WHERE job = 'CLERK');
```

75

# Subqueries

## ◆ Derived relations (inline views)

- 예: 직무가 "CLERK"인 사원이 있는 부서의 이름은?

3. 

```
SELECT DISTINCT dname
 FROM dept,
 (SELECT deptno
 FROM emp
 WHERE job = 'CLERK') d // d: derived relation (inline view)
 WHERE dept.deptno = d.deptno;
```

4. 

```
SELECT DISTINCT dname
 FROM dept d , emp e
 WHERE d.deptno = e.deptno
 AND e.job = 'CLERK';
```

76

# Data Definition Lang. (DDL)

- ◆ Table 생성
  - CREATE TABLE *table명* (*column명 column타입*, ...)
- ◆ Table 구조 변경
  - ALTER TABLE *table명* {ADD, MODIFY, RENAME, DROP} ...
- ◆ Table 이름 변경
  - RENAME *table명* TO *새로운\_table명*
- ◆ Table 삭제
  - DROP TABLE *table명*
- ◆ Table truncate
  - TRUNCATE TABLE *table명*

77

# Data Types

| 자료형의 종류                | 의미                                                                   |
|------------------------|----------------------------------------------------------------------|
| CHAR(N)                | 주어진 크기만큼 고정 길이의 문자 저장<br>1바이트~2000바이트                                |
| VARCHAR2(N)            | 주어진 크기만큼 가변 길이의 문자 저장<br>1바이트~4000바이트<br><i>주의!</i>                  |
| NVARCHAR2(N)           | 국가별 국가 집합에 따른 크기의 문자 또는 바이트의 가변 길이 문자<br>1바이트~4000바이트                |
| NUMBER(p, s)           | 정밀도와 스케일로 표현되는 숫자                                                    |
| DATE                   | 날짜 형식을 저장 (년/월/일)<br><i>주의!</i>                                      |
| ROWID                  | 테이블내 행의 고유 주소를 가지는 64진수 문자<br>해당 6바이트(제한된 ROWID) 또는 10바이트(확장된 ROWID) |
| BLOB                   | 대용량의 바이너리 데이터를 저장<br>최대 4GB                                          |
| CLOB                   | 대용량의 텍스트 데이터를 저장<br>최대 4GB                                           |
| BFILE                  | 대용량의 바이너리 데이터를 파일 형태로 저장<br>최대 4GB                                   |
| TIMESTAMP(n)           | DATE 형의 확장된 형태                                                       |
| INTERVAL YEAR TO MONTH | 년과 월을 이용하여 기간을 저장                                                    |
| INTERVAL DAY TO SECOND | 일, 시, 분, 초를 이용하여 기간을 저장<br>두 날짜 값의 정확한 차이를 표현하는데 유용                  |

78

# Data Types

| Data Type                    | Description                                                                                                                                                                                                                   |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CHAR [(size [BYTE   CHAR])]  | Fixed-length character data of length size bytes or characters. (1 ~ 2000 bytes or characters)                                                                                                                                |
| NCHAR[(size)]                | Fixed-length character data of length size characters. Maximum size is determined by the national character set definition, with an upper limit of 2000 bytes. Default and minimum size is 1 character.                       |
| VARCHAR2(size [BYTE   CHAR]) | Variable-length character string having maximum length size bytes or character s. (1 ~ 4000 bytes or characters) You must specify size for VARCHAR2.                                                                          |
| NVARCHAR2(size)              | Variable-length Unicode character string having maximum length size character s. Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2. |
| LONG                         | Character data of variable length up to 2 gigabytes, or $2^{31}-1$ bytes. Provided for backward compatibility.                                                                                                                |
| NUMBER [(p [ , s ])]         | Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127. A NUMBER value requires from 1 to 22 bytes.                                                             |
| FLOAT [(p)]                  | A subtype of the NUMBER datatype having precision p. A FLOAT value is represented internally as NUMBER. The precision p can range from 1 to 126 binary digits. A FLOAT value requires from 1 to 22 bytes.                     |

79

# Data Types

| Data Type                                                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DATE                                                            | Valid date range from January 1, 4712 BC, to December 31, 9999 AD. The size is fixed at 7 bytes. This datatype contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| TIMESTAMP [(fractional_seconds_precision)]                      | Year, month, and day values of date, as well as hour, minute, and second values of time, where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values of fractional_seconds_precision are 0 to 9. The default is 6. The default format is determined explicitly by the NLS_DATE_FORMAT parameter or implicitly by the NLS_TERRITORY parameter. The sizes varies from 7 to 11 bytes, depending on the precision. This datatype contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It contains fractional seconds but does not have a time zone. |
| INTERVAL YEAR [(year_precision)] TO MONTH                       | Stores a period of time in years and months, where year_precision is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default is 2. The size is fixed at 5 bytes.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds)] | Stores a period of time in days, hours, minutes, and seconds, where day_precision is the maximum number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2. fractional_seconds_precision is the number of digits in the fractional part of the SECOND field. Accepted values are 0 to 9. The default is 6. The size is fixed at 11 bytes.                                                                                                                                                                                                                                                                |

80

# Data Types

| Data Type       | Description                                                                                                                                                                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RAW(size)       | Raw binary data of length size bytes. Maximum size is 2000 bytes. You must specify size for a RAW value.                                                                                                                                                                 |
| LONG RAW        | Raw binary data of variable length up to 2 gigabytes.                                                                                                                                                                                                                    |
| ROWID           | Base 64 string representing the unique address of a row in its table. This datatype is primarily for values returned by the ROWID pseudocolumn.                                                                                                                          |
| UROWID [(size)] | Base 64 string representing the logical address of a row of an index-organized table. The optional size is the size of a column of type UROWID. The maximum size and default is 4000 bytes.                                                                              |
| CLOB            | A character large object containing single-byte or multibyte characters. Both fixed-width and variable-width character sets are supported, both using the database character set. Maximum size is (4 gigabytes - 1) * (database block size).                             |
| NCLOB           | A character large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, both using the database national character set. Maximum size is (4 gigabytes - 1) * (database block size). Stores national character set data. |
| BLOB            | A binary large object. Maximum size is (4 gigabytes - 1) * (database block size).                                                                                                                                                                                        |
| BFILE           | Contains a locator to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. Maximum size is 4 gigabytes.                                                                                     |

81

# Table 생성

## ◆ Subquery를 이용한 table 생성

- 예: emp와 동일한 구조의 table 생성 및 데이터 복사

```
CREATE TABLE emp02
```

AS SELECT \* FROM emp; *AS의 선택과 함께 실행됨!*

```
SELECT * FROM emp02;
```

| EMPNO | ENAME  | JOB       | MGR  | HIREDATE | SAL  | COMM | DEPTNO |
|-------|--------|-----------|------|----------|------|------|--------|
| 7369  | SMITH  | CLERK     | 7902 | 80/12/17 | 800  | -    | 20     |
| 7499  | ALLEN  | SALESMAN  | 7698 | 81/02/20 | 1600 | 300  | 30     |
| 7521  | WARD   | SALESMAN  | 7698 | 81/02/22 | 1250 | 500  | 30     |
| 7566  | JONES  | MANAGER   | 7839 | 81/04/02 | 2975 | -    | 20     |
| 7654  | MARTIN | SALESMAN  | 7698 | 81/09/28 | 1250 | 1400 | 30     |
| 7698  | BLAKE  | MANAGER   | 7839 | 81/05/01 | 2850 | -    | 30     |
| 7782  | CLARK  | MANAGER   | 7839 | 81/06/09 | 2450 | -    | 10     |
| 7839  | KING   | PRESIDENT | -    | 81/11/17 | 5000 | -    | 10     |
| 7844  | TURNER | SALESMAN  | 7698 | 81/09/08 | 1500 | 0    | 30     |
| 7900  | JAMES  | CLERK     | 7698 | 81/12/03 | 950  | -    | 30     |
| 7902  | FORD   | ANALYST   | 7566 | 81/12/03 | 3000 | -    | 20     |
| 7934  | MILLER | CLERK     | 7782 | 82/01/23 | 1300 | -    | 10     |

기존 제약은  
(외래 키 등)  
자동으로  
복사되게 된다!

83

# Table 생성

## ◆ Table 정의

```
CREATE TABLE emp01
```

(empno NUMBER(4), // 4자리 정수 저장 가능  
ename VARCHAR2(20), // 최대 20개의 영문자 저장 가능  
sal NUMBER(7, 2)); // 소수점 위 5자리, 소수점 이하 2자리까지 저장 가능

각체 유형 TABLE 각체 EM01

| Table | Column | 데이터 유형   | 길이 | 전체 자릿수 | 소수점 이하 자릿수 | 기본 키 | 널 가능 | 기본값 | 설명 |
|-------|--------|----------|----|--------|------------|------|------|-----|----|
| EM01  | EMPNO  | Number   | -  | 4      | 0          | -    | ✓    | -   | -  |
|       | ENAME  | Varchar2 | 20 | -      | -          | -    | ✓    | -   | -  |
|       | SAL    | Number   | -  | 7      | 2          | -    | ✓    | -   | -  |

82

# Table 생성

## ◆ Subquery를 이용한 table 생성

- emp에서 일부 column들만 선택하여 table 생성 및 데이터 복사

```
CREATE TABLE emp03
```

AS SELECT empno, ename FROM emp;

- Table 생성 시 column 이름 변경

```
CREATE TABLE emp04 (emp_number, emp_name)
```

AS SELECT empno, ename FROM emp;

- 데이터는 복사하지 않고 table만 동일한 구조로 생성

```
CREATE TABLE emp05
```

AS SELECT \* FROM emp WHERE 1 = 0;

// 조건이 항상 거짓이므로 선택되는 데이터가 없음

84

# Table 변경

## ◆ Column 추가

```
ALTER TABLE emp01
```

```
ADD (email VARCHAR2(10), age NUMBER(2));
```

- 기존 column을 다음에 추가됨

일부에 한해  
타입 변경도 가능!  
(보통된다면)

## ◆ Column 변경 (타입, 크기, 제약조건, 기본값 등)

```
ALTER TABLE emp01
```

```
MODIFY (email VARCHAR2(40), age CHAR(2));
```

- Table의 column에 데이터가 존재하는 경우 대부분 타입 변경 불가능
  - ✓ CHAR와 VARCHAR2 사이의 타입 변경 가능
  - ✓ 기존 타입보다 크기가 증가하는 경우에 한해서 크기 변경 가능

85

# Table 변경

## ◆ Column 이름 변경

```
ALTER TABLE emp01
```

```
RENAME COLUMN ename TO emp_name;
```

## ◆ Column 삭제

```
ALTER TABLE emp01
```

```
DROP COLUMN email;
```

- Table의 column에 데이터가 존재하는 경우 모두 삭제됨

## ◆ Table 이름 변경

```
RENAME emp01 TO emp_01;
```

86

# Table 삭제

## ◆ Table 및 데이터 삭제

```
DROP TABLE dept;
```

- Table과 관련된 모든 index, 제약조건, trigger, 권한 등도 같이 삭제됨
- 삭제하려는 table을 다른 table에서 foreign key로 참조하고 있는 경우 삭제 불가

ORA-02449: 외래 키에 의해 참조되는 고유/기본 키가 테이블에 있습니다

```
DROP TABLE dept CASCADE CONSTRAINTS;
```

- 다른 테이블에서 dept를 참조하고 있는 foreign key 제약조건이 있을 경우 그것들을 먼저 삭제하고 table 삭제 실행
- dept를 참조하는 테이블(예: emp)의 행들은 삭제되지 않음

87

# Table Truncate

## ◆ Table 자체는 삭제하지 않고 데이터만 삭제

```
TRUNCATE TABLE dept;
```

- Table에서 모든 행을 삭제하는 가장 빠르고 효율적인 방법
- Table과 관련된 구조(제약조건, trigger 등)와 권한에 영향을 주지 않음
- Table에 연관되어 있는 trigger가 실행되지 않음
- 즉시 commit하므로 rollback될 수 없음

## - 비교

- **DELETE FROM dept;**
- Table 삭제(drop) 후 재생성(create)

88

# Data Manipulation Lang. (DML)

## ◆ Table에 새로운 데이터(행) 삽입

- `INSERT INTO table명 [(column-list)] VALUES (value-list)`
- `INSERT INTO table명 [(column-list)] Subquery`

## ◆ Table에 저장된 데이터(column 값) 수정

- `UPDATE table명 SET column명=값, column명=값 ... [WHERE 조건절]`
- `UPDATE table명 SET (column-list) = (Subquery) [WHERE 조건절]`

## ◆ Table에 저장된 데이터(행) 삭제

- `DELETE FROM table명 [WHERE 조건절]`

89

# 데이터 삽입

## ◆ Column 값을 지정하여 하나의 행을 삽입

```
INSERT INTO dept (deptno, dname, loc)
VALUES (60, '회계과', '서울');
```

### - Column-list와 value-list의 길이가 일치하지 않으면 오류 발생

```
INSERT INTO dept (deptno, dname, loc) // error!
VALUES (60, '회계과');
```

### - Column 이름이나 값의 타입이 올바르지 않으면 오류 발생

```
INSERT INTO dept (deptnum, dname, loc) // error!
VALUES (60, '회계과', 서울);
```

# 데이터 삽입

## - 모든 column 값을 입력하는 경우에는 column-list 생략 가능

```
INSERT INTO dept
VALUES (60, '회계과', '서울');
```

## - Column-list와 value-list에 모두 누락된 column에 대해서는 NULL 값 저장

```
INSERT INTO dept (deptno, dname)
VALUES (60, '회계과'); // loc에는 NULL 값 저장
```

- 누락된 column에 NOT NULL 제약조건이 있을 경우 오류 발생
- 다음과 동일

```
INSERT INTO dept (deptno, dname, loc)
VALUES (60, '회계과', NULL);
```

```
INSERT INTO dept (deptno, dname, loc)
VALUES (60, '회계과', ""); // 주의: ''(공백문자) 아님
```

91

# 데이터 삽입

## ◆ Subquery를 사용하여 입력될 데이터 생성

```
INSERT INTO dept // VALUES 생략
SELECT * FROM old_dept;
```

### - Subquery에 의해 생성되는 column의 타입들이 table의 column 타입들과 호환 가능해야 함

- column 이름은 일치하지 않아도 됨

```
INSERT INTO dept01 (deptno, dname) // loc을 생략할 경우
SELECT deptnum, 'ACC'
FROM dept
WHERE loc='BOSTON';
```

- 모든 dname에는 'ACC', loc에는 NULL 값이 저장됨

값이 col

92

# 데이터 수정

- ◆ 변경할 column에 대해 값 지정

```
UPDATE dept
SET dname='Production', loc='서울'
WHERE deptno=10;
```

- deptno 값이 10인 행들에 대해 dname과 loc 값을 지정된 값으로 변경

```
UPDATE emp
SET sal = sal * 1.1;
▪ 모든 사원의 급여를 10% 인상
```

93

# 데이터 수정

- ◆ Subquery를 이용하여 변경할 값 생성

```
UPDATE dept
SET (dname, loc) = (SELECT dname, loc
FROM dept
WHERE deptno=40)
WHERE deptno=20;
```

- 부서번호가 20인 부서의 부서명과 지역을 부서번호가 40인 부서의 부서명과 지역으로 변경

# 데이터 삭제

- ◆ 조건을 만족하는 행들을 table에서 삭제

```
DELETE FROM dept
WHERE deptno=30;
```

- ◆ 조건절에서 subquery 사용

```
DELETE FROM emp
WHERE deptno = (SELECT deptno
FROM dept
WHERE dname='SALES');
```

- SALES 부서에 속한 모든 사원들의 행들을 삭제

95

# 데이터 병합

- ◆ Table의 데이터를 다른 table에 병합
  - 새로운 행들을 삽입 또는 기존 행을 변경

```
MERGE INTO department n
USING dept d
ON (n.deptno = d.deptno)
WHEN MATCHED THEN
 UPDATE SET
 n.dname = d.dname,
 n.loc = d.loc
WHEN NOT MATCHED THEN
 INSERT VALUES (d.deptno, d.dname, d.loc);
```

- "dept" table의 모든 행들을 "department" table에 삽입하되, 동일한 deptno 값이 department에 이미 존재하면 그 행의 column 값을 dept 행의 값들로 수정함

94

96

# References

---

- ◆ Online Documentation
  - SQL Language Quick Reference
    - <https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlqr/index.html>
  - SQL Language Reference
    - <https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/>
  - Oracle에서 제공되는 functions
    - <https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/Functions.html#GUID-D079EFD3-C683-441F-977E-2C9503089982>
    - <https://www.techonthenet.com/oracle/functions/>
- ◆ Books
  - 이상구 외 3, 데이터베이스의 이해, 이한미디어, 2012.
  - 박우창, 남송희, 이현룡, 오라클로 배우는 데이터베이스 개론과 실습, 2판, 한빛아카데미, 2020.
  - 성윤정, Oracle 11g 프로그래밍, 북스홀릭, 2011.



