

① (18/11/10)

<SQL> 시작~

• RDBMS : Relational Database Management System

• 데이터 모델

 [계층형] hierarchical

 [네트워크형] network

 [관계형] 가장 많아요

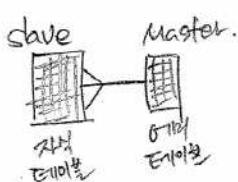
 [객체 지향형] : (프로그래밍 언어들)
Object-oriented

• 관계형 데이터 베이스 구성 요소

- SQL : Structured Query Language

- row, column

- column, key



- 기본키 primary key
- 후보키 candidate key
- 보조키 alternate key
- 외래키 foreign key
- 복합키 composite key

• 오라클 DB 버전 : 11g 버전 (현재 가장 많은)

• 자료형 (data type)

 1) VARCHAR2 (길이)

 2) NUMBER (정밀자리수, 소수점자리수)

 3) DATE

* 영문 : 1 Byte / 한글

한글 : 2 Byte / 한글

• 데이터 조회 방법

- 셀렉션 : 행 단위
- 프로젝션 : 열 단위
- 조인 : 두 개 이상 테이블
 마치 하나의 테이블
 (외래키)

• SELECT 열 이름 1, 열 이름 2, ...

• FROM 테이블 이름 ;

• DISTINCT (열 중복제거문)

- SELECT DISTINCT 열 이름 1, 2, ...

- " " ALL " "

• 열과 연산식

- SELECT 열 이름 1, 2, ..., 열 X 1/2 + 열 3, 열 3

* NULL : 값이 존재하지 않는다는 뜻.

• 별칭을 지정하는 방식

1) 열 이름 별칭

2) " " "별칭"

3) " AS 별칭" 가장 많이 쓰고 있음

4) " AS "별칭"

• 오름차순 & 내림차순 => ORDER BY

ASC DESC

(기본 설정)

- SELECT * FROM EMP

ORDER BY 열 이름 1;

- SELECT * FROM EMP

ORDER BY 열 이름 1 DESC;

* 동시 적용 ORDER BY 열 1 ASC, 열 2 DESC;

② (18/11)

< WHERE 절과 연산자 >

• WHERE 열 이름 연산자

열 이름 = 30

열 이름 = '문자'

대소문자 구별 ✓

• AND, OR 연산자

• 산술연산자, 비교연산자

* WHERE 열 이름 \geq 'F';

그 열의 첫문자와 대용자 F를 비교했을 때
알파벳 순서상 F와 같거나
F보다 뒤에 있는 문자열을 출력하라

WHERE 열 이름 \leq 'FORZ';

F, O, R, Z 순으로 확인

- \neq , $\langle \rangle$, \wedge , \neg , NOT

많이 쓴다.

• IN 연산자

WHERE JOB IN ('열데이터', '데이터');

문자일 때

문자일 때는 숫자만

• BETWEEN A AND B 연산자

WHERE 열 이름 BETWEEN 하한 AND 한한

• LIKE 연산자 & 와일드 카드
(%, -)

WHERE 열 이름 LIKE '%L%';

어떤 어떤 상관없이
한개의 문자 데이터 의미

같이 상관없이 (문자 없는 경우 포함)
모든 문자 데이터를 의미

* ESCAPE 절

WHERE 열 이름 LIKE 'A\%A%' ESCAPE '\'

문자로 인식
(와일드카드가 아닐 때)

* LIKE를 잘 활용해야 주의, 검색등 속도 빠름.

• IS NULL 연산자

WHERE 열 이름 = NULL; (X)

" " IS NULL; (O)

IS NOT

AND	T	F	NULL
T	T	F	NULL
F	F	F	F ✓
NULL	NULL	F ✓	NULL

OR	T	F	NULL
T	T	T	T
F	T	F	NULL ✓
NULL	T	NULL ✓	NULL

• 집합연산자 (합집합, 차집합, 교집합)

- UNION 합집합, 중복제거

- UNION ALL 합집합, 중복 모두 포함

- MINUS 차집합

- INTERSECT 교집합

SELECT ~

UNION

SELECT ~

~

* A와 B 열들은
순서와 차집합, 이중이
같아야 한다.

* 연산자 우선순위 : 산술 > 비교 > IS NULL > BETWEEN > NOT > AND OR

③ C18||12)

〈오라클 함수〉

* 함수의 종류 < 내장함수 (built-in function)
 사용자정의함수 (user-defined ~)

→ 단일행 칸수 →

v INITCAP(여기)
_____ 알맞은 자리를 차내에서 고를자

• LENGTH 함수

✓ SELECT 열이름, LENGTH(열이름)
문자열 길이(개수) 출력

* LENGTHB : 바이트수 -하기 값수.

* DUAL 테이블 : dummy 테이블

↳ 임시 연산이나 함수 결과 값 확인용도 사용

• SUBSTR 함수 : 문자열 일부 추출 (135P)

✓ SUB(JOB, 1, 2) → SALES MAN
 의미 의미
 ↓
 마지막에 없으면 끝까지 의미

◦ TNSTR 함수 : 문자열 데이터에서 문자 위치 찾기

(137P) ✓ INSTR ('^{설정값}문자열', '문자', 시작위치, 끝위치)
 ✓ INSTR ('HELLO', 'L', 1, 2)

선택

✓ TSTRUCT(ENAME, 'S')

卷之三

✓ REPLACE 함수 : 문자 교체

✓ REPLACE ('010-1234-5678', '-')
↳ 01012345678

- o LPAD, RPAD 함수 : 빈공간 채우기
Left Padding Right Padding

✓ LPAD ('ABC', 5, '#') 선택

✓ RPAD('ABC', 6, '*')
 ↑
 |B ABC*****

* RPAD ('ABC', 6) \Rightarrow ABC||| \rightarrow 8位.

⑨ CONCAT 함수 : 하나로 합치는

✓ CONCAT('A', 'B') → 'A' || 'B'

* 하나로 합치는 또 다른 함수 || 연산자

- TRIM, LTRIM, RTRIM 함수 (143P)
◦ (특정 문자를 제거하는 함수)

✓ TRIM (선택)

 ↗ LEADING (앞쪽부터)

 ↗ TRAILING (오른쪽부터)

 ↗ BOTH (앞쪽, 오른쪽 다)

 ↗ 선택

↗ 문자 (선택)

 ↗ FROM '문자열'

 ↗ 삭제할

 ↗ 디코드

↗ 있는 경우: 문자 지운다.

 ↗ 없는 경우: 공백만 지운다.

✓ LTRIM ('_Oracle_'), '_<')
↳ Oracle >

✓ RTRIM('<_Oracle_>', '>_')
↳ / Oracle

④ (18/11/13)

<숫자 처리 함수>

• ROUND 함수 : 반올림(숫자)

✓ ROUND (1234.5678, 0) \Rightarrow 1235

" (1234.5678, 1) \Rightarrow 1234.6

" (" , -1) \Rightarrow 1230

" (" , -2) \Rightarrow 1200

선택 \rightarrow 없으면 소수점자리 반올림

• TRUNC 함수 : 내림(숫자)

✓ TRUNC (1234.5678, 0) \Rightarrow 1234
선택

• CEIL, FLOOR 함수

가장 큰
가장 작은

✓ CEIL (3.14), FLOOR (3.14), CEIL (-3.14), FC
 \downarrow
4
 \downarrow
3
 \downarrow
-3
 \downarrow
-4

• MOD 함수 : 숫자를 나눈 나머지 값 구하기

✓ MOD (15,6), MOD (10,2), MOD (11,2)

\downarrow
3
 \downarrow
0
 \downarrow
1

<날짜 처리 함수>

• ADD_MONTHS 함수 : 몇 개월 이후 날짜

✓ ADD_MONTHS (SYSDATE, 3)

\downarrow 현재날짜
2018-10-13 오전 11:49:32
+개월수

• MONTHS_BETWEEN 함수 : 두 날짜 개월수 차

✓ MONTHS_BETWEEN (SYSDATE, HIREDATE)

\downarrow
441.386964979092

* TRUNC 함수 사용해서 정수만들자

TRUNC () \Rightarrow 441

• NEXT_DAY, LAST_DAY 함수

돌아오는 요일

단위 마지막 날짜

✓ NEXT_DAY (SYSDATE, '월요일')

\downarrow
7/13

2018-07-16 오후 11:55:11

✓ LAST_DAY (SYSDATE)

\downarrow
7/13

2018-07-31 오후 11:55:11

• ROUND, TRUNC 함수 (날짜)

반올림
(155P)

\Rightarrow 2018-07-13 오후 11:56:28
반세기
→ 2001-01-01

✓ ROUND (SYSDATE, 'CC')
→ 2001-01-01
↑ 반년

" (" , 'YY') \Rightarrow 2019-01-01
↑ 분기별

" (" , 'Q') \Rightarrow 2018-07-01
↑ 3분기

" (" , 'DDD') \Rightarrow 2018-07-14
↑ 30일

" (" , 'HH') \Rightarrow 2018-07-14
↑ 30분

✓ TRUNC \rightarrow 동일 Type

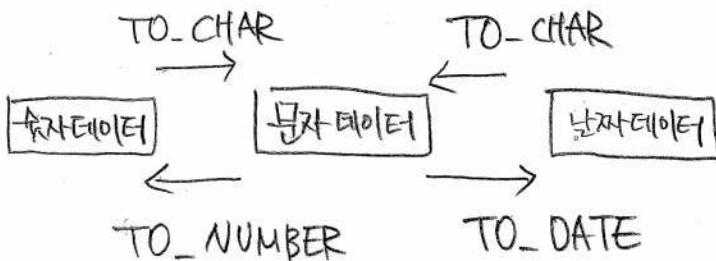
<자료형 변환 함수>

* 자료형 변환 방법
 : 암시적 형변환 (= 자동형변환)

 : implicit type conversion

명시적 형변환 (= 직접변환)

 : explicit type conversion



④ (18/11/13)

• TO_CHAR 함수 (\rightarrow 숫자 \leftarrow 문자) (159P)

• TO_CHAR (SYSDATE, 'YYYY/MM/DD')

\downarrow HH24:MI:SS')

2018/07/13 23:59:01

• TO_CHAR (SYSDATE, 'MM') \Rightarrow 07

" (" , 'MON') \Rightarrow 7월

" (" , 'MONTH') \Rightarrow 7월

" (" , 'DD') \Rightarrow 14

" (" , 'DT') \Rightarrow 토

" (" , 'DAY') \Rightarrow 토요일

• TO_CHAR (SYSDATE, 'MON', 'NLS_DATE_LANGUAGE=')



* 시간형식 { HH24

HH, HH12

MI

SS

AM, PM, A.M., P.M.

(숫자 데이터 형식 지정)

TO_CHAR (SAL, '\$ 999,999') \Rightarrow \$800

" (" , '\$ 999,999') \Rightarrow \$800

" (" , '999,999.00') \Rightarrow 800.00

" (" , '000,999,999.00') \Rightarrow 1,800.00

" (" , '000,999,999.99') \Rightarrow 1,800.00

" (" , '999,999,00') \Rightarrow 800

변환 (163P)

• TO_NUMBER 함수 (- 숫자 문자 날짜)

• TO_NUMBER ('1,300', '999,999')

\downarrow 숫자 * 세미콜론 때문에 연산처리 불필
1,300
숫자

↳ 자동 숫자 변환 안됨.

변환

• TO_DATE 함수 (\rightarrow 숫자 \leftarrow 날짜)

• TO_DATE ('2018-07-14', 'YYYY-MM-DD')

" (" , ' ')

• WHERE TO_DATE (' ', 'YYYY/MM/DD');
HIREDATE >

부등호 (비교연산자)

• TO_DATE ('49/12/10', 'YY/MM/DD') \Rightarrow 2049/12/10

49 RR \Rightarrow 2049/12/10

50 FF \Rightarrow 2050/12/10

50 RR \Rightarrow 1950/12/10

51 FF \Rightarrow 2051/12/10

51 RR \Rightarrow 1951/12/10

* RR { 0~49 \Rightarrow 2000 ~ 2049

50~99 \Rightarrow 1950 ~ 1999

< NVL 처리함수 >

• NVL 함수

• NVL (COMM, 0) \rightarrow NULL인 경우 반환 데이터
열 데이터

NULL이 아닐 경우.

• NVL2 함수 \uparrow \rightarrow NULL인 경우

• NVL2 (COMM, '0', 'X')

" (COMM, SAL*12+COMM, SAL*12) \rightarrow NULL인 경우
NULL아닐경우 NULL인 경우

⑥ (18/11/13 ~ 18/11/14)

< 상황에 따라 다른 데이터 반환 >

◦ DECODE 함수 : N 조건문과 유사

◦ DECODE (JOB, 불린 데이터

'MANAGER', SAL*1.1,

'SALESMAN', SAL*1.05,

SAL*1.03)

↳ else 반환값.

* ◦ CASE 문 CASE문 > DECODE문

◦ CASE (JOB) 기준 데이터

WHEN 'MANAGER' THEN SAL*1.1

WHEN 'SALESMAN' THEN SAL*1.05

ELSE SAL*1.03

END

(기준데이터 없이도 가능)

◦ CASE
WHEN COMM IS NULL THEN '배당없음'
WHEN COMM = 0 THEN '수당없음'
WHEN COMM > 0 THEN '수당'

END

↳ 단일행 함수 끝

다음부터는 다중행 함수.

* ◦ 단일행 함수 참고 사이트. (173 p)

< 다중행 함수와 데이터 그룹화 >

multiple-row function.

(= 그룹함수 = 복수행 함수)

◦ SUM(SAL) = SUM(ALL SAL)

◦ SUM(DISTINCT SAL)

↳ 중복 데이터는 제외하고 합계를 한다.

◦ COUNT 함수

- SELECT COUNT(*)

FROM EMP;

COUNT(*)
14

- COUNT(DISTINCT SAL)

↳ 중복 데이터 제거

◦ MAX, MIN 함수

- MAX(SAL)

- MIN(SAL)

◦ AVG 함수 : 평균 구하기

- AVG(DISTINCT SAL)

↳ 중복 데이터 제거, 선택

◦ GROUP BY 절

- SELECT AVG(SAL), DEPTNO
FROM EMP
GROUP BY DEPTNO;

AVG(SAL)	DEPTNO
1566.666	30
2125	20
2916.666	10

↳ SELECT 절의 모든 단일행

이 적용해야 한다.

(왜냐하면 다중행

AVG(SAL)와 how

해야 다른 대상에

적용가능하다)

⑦ (18/11/15)

- HAVING 절 : GROUP BY 절에 사용하는 WHERE과 같은 기능

```

SELECT DEPTNO, JOB, AVG(SAL)
FROM EMP
GROUP BY DEPTNO, JOB
HAVING AVG(SAL) >= 2000
ORDER BY DEPTNO, JOB;
    
```

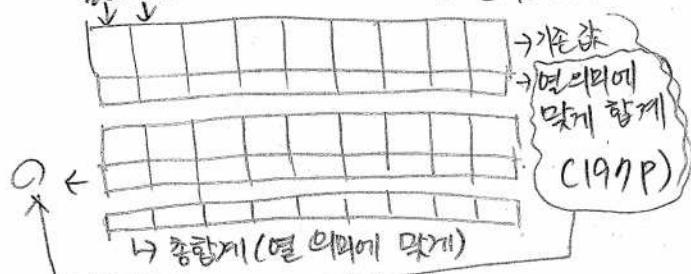
▶ 무언가 GROUP BY 절 다음에 올수 있다.

※ 항상 WHERE 절이 같은 SELECT 문에 있으면 WHERE 절 실행 후 HAVING 절 실행

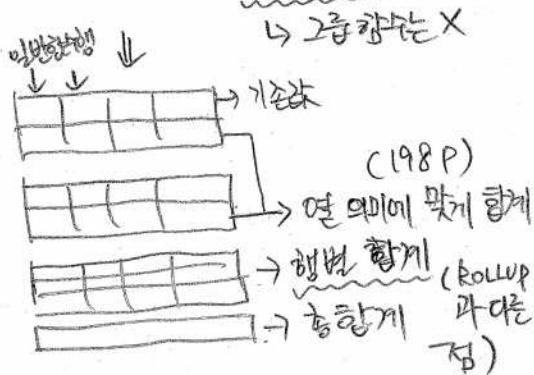
< 그룹화(다중행 함수)와 관련된 이의 함수>
↳ (실무 사용빈도 ↓)

ROLLUP, CUBE, GROUPING SETS 함수
함께 함께 별도로 고급으로 끌어올려.

- GROUP BY ROLLUP (DEPTNO, JOB)
 ↳ 행단위합계 ↳ 그룹화하는 X



- GROUP BY CUBE (DEPTNO, JOB)
 ↳ 행단위합계 ↳ 그룹화하는 X



- GROUP BY GROUPING SETS (DEPTNO, JOB)

DEPTNO	JOB	COUNT (*)
10	CLERK	3
20	SALESMAN	5
30	ANALYST	6
	2 3 4

◦ GROUPING 함수 : 그룹화 함수 (203P)

- GROUPING (DEPTNO)

◦ GROUPING_ID 함수 : 그룹화 함수 (205P)

◦ LISTAGG 함수 : 행열을 가로로 출력

(외로 "용여터 설정")

- SELECT DEPTNO,

LISTAGG (ENAME, ',')

WITHIN GROUP (ORDER BY SAL DESC)
AS ENAMES

FROM EMP

GROUP BY DEPTNO;

DEPTNO	ENAMES
10	KING, CLARK, MILLER
20	FORD, SCOTT, JONES, ADAMS
30	BLAKE, ALLEN, TURNER, ...

◦ PIVOT, UNPIVOT 함수 (208P)
(행→열) (열→행)

- SELECT *

FROM (~)

PIVOT (MAX(SAL))

FOR DEPTNO IN (10, 20, 30)

ORDER BY JOB;

출력 DATA

가로줄 표기법.

열 DATA 작성

* UNPIVOT (SAL FOR JOB IN (CLERK, ...))
(211P)

출력 DATA

세로줄

행 DATA 작성

8) (18/11/15)

< 조인 >

- 조인하기 (등가조인)

- SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
ORDER BY EMPNO;

- 조인 + 별칭 설정

- SELECT E.EMPNO, E.ENAME, E.JOB, E.MGR,
E.HIREDATE, E.SAL, E.COMM,
E.DEPTNO, D.DNAME, D.LOC
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
ORDER BY EMPNO;

↳ 열 이름이 두 테이블에서 같으므로 → 복수개
위해 필요하며 이 외 다른 열은 굳이 필요없음.

※ WHERE 절에 추가로 조건 넣어 출력하기.
- FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND SAL >= 3000;

= 단순조인 (Simple Join)
= 내부조인 (Inner Join)

• 조인의 종류

- 등가 조인 (Equi Join)
 - 비등가 조인 (Non-equi Join)
 - 자체 조인 (Self Join)
 - 외부 조인 (Outer Join)

• 비등가 조인 (조인 테이블간 일치 열이 없을 때)
(224p)

- SELECT *
FROM EMP E, SALGRADE S
WHERE E.SAL BETWEEN S.LOSAL
AND S.HISAL;

○ 자체 조인 (Self join) 같은 테이블 내 같은 열이
같은 테이블을 두 번 사용하여 자체 조인하기)

- SELECT E1.EMPNO, E1.ENAME, E1.MGR,
E2.EMPNO AS MGR_EMPNO,
E2.ENAME AS MGR_ENAME
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO;

- 외부 조인 (Outer join)

(위 자체 조인식 출력 결과 NULL값 발생시
행 삭제 발생되나 강제로 결합 가능)
즉, 자체 조인은 데이터가 존재 할 때만 출력
외부 조인은 NULL 값으로 강제 출력한다.

1) 왼쪽 외부 조인 (229p)

SELECT ~

FROM EMP E1, EMP E2

WHERE E1.MGR = E2.EMPNO(+)

ORDER BY E1.EMPNO;

2) 오른쪽 외부 조인 (229p)

SELECT ~

FROM EMP E1, EMP E2

WHERE E1.MGR(+)=E2.EMPNO

ORDER BY E1.EMPNO;

Left outer join



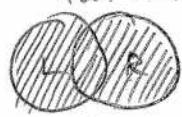
Right outer join



inner join



full outer join



⑨ (18/11/15)

<SQL-99 표준 문법으로 배우는 조인>

◦ NATURAL JOIN (등가조인 대신 사용) ~ 일반등가조인

- SELECT E.EMPNO, E.ENAME, E.JOB,
 E.MGR, E.HIREDATE, E.SAL,
 E.COMM, DEPTNO, D.DNAME
 D.LOC
 ↳ 테이블 이름 붙이면 안됨.

FROM EMP E NATURAL JOIN DEPTD
ORDER BY DEPTNO, E.EMPNO;

◦ JOIN ~ USING (기존 등가조인 대신 사용)

- SELECT ~, DEPTNO, D.DNAME, ~
 FROM EMP E JOIN DEPT D USING
 (DeptNo)
 WHERE SAL >= 3000
 ORDER BY DEPTNO, E.EMPNO;

사용가능

※ ◦ JOIN ~ ON (기존 등가조인 대신 사용)

↳ 가장 범용성 있는
 ↳ 테이블 이름
- SELECT ~, E.DEPTNO, ~
 FROM EMP E JOIN DEPT D ON
 (E.DEPTNO = D.DEPTNO)
 WHERE SAL <= 3000
 ORDER BY E.DEPTNO, EMPNO;

사용하는 개별자

나옴.

◦ OUTER JOIN (SQL-99 표준)

1) 왼쪽 외부 조인 (SQL-99)

- SELECT E1.EMPNO, E1.ENAME, E1.MGR,
 E2.EMPNO AS MGR-EMPNO,
 E2.ENAME AS MGR-ENAME
 FROM EMP E1 LEFT OUTER JOIN EMP E2 ON
 (E1.MGR = E2.EMPNO)
 ORDER BY E1.EMPNO;

2) 오른쪽 외부 조인 (SQL-99)

- SELECT ~
 FROM EMP E1 RIGHT OUTER JOIN EMP E2 ON
 (E1.MGR = E2.EMPNO)
 ORDER BY E1.EMPNO;

3) 전체 외부조인 (SQL-99) full outer join

- SELECT E1.EMPNO, E1.ENAME, E1.MGR,
 E2.EMPNO AS MGR-EMPNO,
 E2.ENAME AS MGR-ENAME
 FROM EMP E1 FULL OUTER JOIN EMP E2 ON
 (E1.MGR = E2.EMPNO)
 ORDER BY E1.EMPNO;

◦ 세 개 이상의 테이블 조인 (SQL-99)

- FROM TABLE1 JOIN TABLE2 ON (조건식)
 JOIN TABLE3 ON (조건식)

⑩ (18/11/16)

< 서브쿼리 > ↳ 실무에서 자주 사용
반드시 숙지.

메인쿼리 (main query) > 서브쿼리 (subquery)

- SELECT *

```
FROM EMP
WHERE SAL > (SELECT SAL
    FROM EMP
    WHERE ENAME = 'JONES'))
```

< 실행 결과가 하나인 단일행 서브쿼리>
Single-row subquery

• 단일행 서브쿼리와 날짜형 테이터

- SELECT *
 FROM EMP
 WHERE HIREDATE < (SELECT HIREDATE
 FROM EMP
 WHERE ENAME = 'SCOTT'))

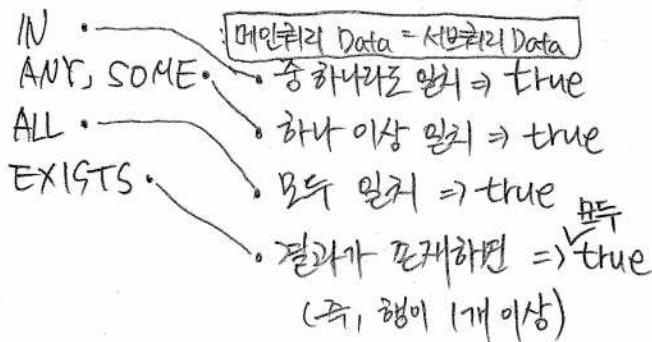
• 단일행 서브쿼리와 조인

- SELECT E.EMPNO, E.ENAME, ~ D.LOC
 FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
 AND E.DEPTNO = 20
 AND E.SAL > (SELECT AVG(SAL)
 FROM EMP))

* 조인과 서브쿼리를 함께 사용해서 많이 쓴다.

< 실행 결과가 여러 개인 다중행 서브쿼리>
multiple-row subquery

※ 다중행 연산자



- SELECT *
 FROM EMP
 WHERE SAL IN (SELECT MAX(SAL)
 FROM EMP
 GROUP BY DEPTNO);

- SELECT *
 FROM EMP
 WHERE SAL = ANY (SELECT MAX(SAL)
 FROM EMP
 GROUP BY DEPTNO);

↑ 같은 결과값.
→ 예: ANY, SOME 연산자를
등가비교 연산자 (=)와 함께 사용하면
IN 연산자와 정확히 같은 기능 수행

- SELECT *
 FROM EMP
 WHERE SAL < ANY (SELECT SAL
 FROM EMP
 WHERE DEPTNO = 30)
 ORDER BY SAL, EMPNO;

의미: 적어도 하나 이상 일치 \Rightarrow true.
↳ 제일 큰 수 그 다음 몇몇 행의
값이 포함되어 있음을 알 수 있다.

즉, < ANY 연산자는
서브쿼리 값 중 항목의 최대값
보다 작은 값을 모두 출력.

SAL
1600
1200
2850 ✓
1400
950

• ALL 연산자

- SELECT *
 FROM EMP
 WHERE SAL < ALL (SELECT SAL
 FROM EMP
 WHERE DEPTNO = 30);

의미: 모두 일치 \Rightarrow true.

• EXISTS 연산자

- SELECT *
 FROM EMP
 WHERE EXISTS (SELECT DNAME
 FROM DEPT
 WHERE DEPTNO = 10);

의미: 하나라도 존재하면 \Rightarrow true
하나도 없으면 BEFORE (부른다)
↳ (존재하는) 빈 테이블

⑪ (18/11/6)

<비교할 열이 여러 개인 다중열 서브쿼리>

multiple-column subquery
= 복수열 서브쿼리

↳ 서브쿼리가 비교할 열 여러개.

- SELECT *

FROM EMP

WHERE (DEPTNO, SAL) IN (SELECT DEPTNO, MAX(SAL)
FROM EMP
GROUP BY DEPTNO);

<FROM 절에 사용하는 서브쿼리와 WITH 절>

◦ 인라인 뷰 : FROM 절에 사용하는 서브쿼리
(inline view)

- SELECT E10.EMPNO, E10.ENAME,
D.DNAME, D.LOC

FROM (SELECT * FROM EMP WHERE DEPTNO=10) E10
(SELECT * FROM DEPT) D

WHERE E10.DEPTNO = D.DEPTNO;

↳ 불필요한 열이 많아 일복행과 열만
사용하고자 할 때 유용하다.

◦ WITH 절을 사용한 서브쿼리

- WITH

E10 AS (SELECT * FROM EMP WHERE
별칭 DEPTNO=10)

D AS (SELECT * FROM DEPT)

SELECT E10.EMPNO, E10.ENAME, E10.DEPTNO,
D.DNAME, D.LOC
FROM E10, D
WHERE E10.DEPTNO = D.DEPTNO;

↳ 여러개의 서브쿼리가 몇 번, 몇 번을 이상
넘나드는 경우와 하위문 경우,
메인쿼리, 서브쿼리 복잡할 때 꽤 유용한.

<SELECT 절에 사용하는 서브쿼리>

↳ 스칼라쿼리 (scalar subquery)

- SELECT EMPNO, ENAME, JOB, SAL,
(SELECT GRADE

FROM SALGRADE

WHERE E.SAL BETWEEN L0SAL AND
H1SAL) AS SALGRADE

DEPTO,

(SELECT DNAME

FROM DEPT

WHERE E.DEPTNO = DEPT.DEPTNO
AS DNAME

FROM EMP;

↳ SELECT 절에 등장하는 서브쿼리는
별도로 결과만 반환하도록 작성해
주어야 한다.

결과
값을

(12) (18/11/17)

<테이블에 데이터 추가하기>

◦ 테이블 생성하기 (복사)

- CREATE TABLE DEPT_TEMP
AS SELECT * FROM DEPT;

◦ 테이블 삭제하기

- DROP TABLE 테이블 이름;

◦ 테이블에 데이터를 추가하는 INSERT문

- INSERT INTO DEPT_TEMP(DEPTNO,DNAME,LOC)
VALUES(50,'DATABASE','SEOUL');
SELECT * FROM DEPT_TEMP;

↓

DEPTNO	DNAME	LOC
~	~	~
~	~	~
50	DATABASE	SEOUL

추가

* 열지정 없이 데이터 추가

- INSERT INTO DEPT_TEMP
VALUES(60,'NETWORK','BUSAN');
SELECT * FROM DEPT_TEMP;

◦ 테이블에 NULL 데이터 입력하기

- INSERT INTO DEPT_TEMP(DEPTNO,DNAME,LOC)
VALUES(70,'WEB',NULL);

▶▶▶
NULL입력

OR(다음방법)

VALUES(70,'WEB','');

- INSERT INTO DEPT_TEMP(DEPTNO,LOC)

▶▶▶ - VALUES(90,'INCHEON');

▶▶▶ NULL입력
자동으로 DNAME열은 NULL입력

데이터 조작어
DML
(Data Manipulation Language)

INSERT
UPDATE
DELETE

◦ 테이블에 날짜 데이터 입력하기

(행 값 없이 열구조만 Table 복사)

- CREATE TABLE EMP_TEMP

AS SELECT *
FROM EMP
WHERE 1 <> 1; (272P)

(날짜 데이터)

- INSERT INTO EMP_TEMP (~)
VALUES(9999,'홍길동','PRESIDENT',NULL,'2001/01/01',
5000,1000,10);

YYYY/MM/DD

YYYY-MM-DD

(TO_DATE 함수 사용 날짜데이터 입력) SYSDATE

- INSERT INTO EMP_TEMP (~)
VALUES(211,'이순신','MANAGER',9999,
TO_DATE('07/01/2001','DD/MM/YYYY'),
4000,NULL,20);

(시브리리로 한번에 여러 데이터 추가)

- INSERT INTO EMP_TEMP (EMPNO,ENAME,JOB,
MGR,HIREDATE,SAL,
COMM,DEPTNO)

SELECT E.EMPNO, E.ENAME, E.JOB,
E.MGR, E.HIREDATE, E.SAL,
E.COMM, E.DEPTNO

FROM EMP E, SALGRADE S

WHERE E.SAL BETWEEN S.LOSAL AND
S.HISAL

AND S.GRADE = 1;

※ INSERT문에 시브리리 사용의 위험점

1) VALUES문을 사용하지 않는다

2) 추가, 업데이트 시브리리 열수 동일
가는 테이블의

3) 기본 테이블 자료형 시브리리 자료형 동일

(13) (18/11/17)

< 테이블에 있는 데이터 수정하기 >

(열 데이터 전체 수정하기)

- UPDATE DEPT TEMP2

SET LOC = 'SEOUL';

(수정한 내용을 되돌리고 싶을 때)

- ROLLBACK; (명령어)

↳ UPDATE 명령어 실행 이전 상태로

(데이터 일부분만 수정하기)

- UPDATE DEPT TEMP2

SET DNAME = 'DATABASE',
LOC = 'SEOUL'

WHERE DEPTNO = 40;

< 서브쿼리를 사용하여 데이터 수정 >

(여러 열을 한 번에 수정하는 경우)

- UPDATE DEPT TEMP2

SET (DNAME, LOC) = (SELECT DNAME,
LOC
FROM DEPT
WHERE DEPTNO=40)

WHERE DEPTNO=40;

(열 하나하나를 수정하는 경우)

- UPDATE DEPT TEMP2

SET DNAME = (SELECT DNAME
FROM DEPT
WHERE DEPTNO=40),
LOC = (SELECT LOC
FROM DEPT
WHERE DEPTNO=40)

WHERE DEPTNO = 40;

(WHERE 절에 서브쿼리를 사용하여 데이터 수정)

- UPDATE DEPT TEMP2

SET LOC = 'SEOUL'

WHERE DEPTNO = (SELECT DEPTNO

FROM DEPT TEMP2
WHERE DNAME = 'OPERATION');

< 테이블에 있는 데이터 삭제하기 >

(데이터 일부분만 삭제하기)

- DELETE FROM EMP TEMP2

WHERE JOB = 'MANAGER';

(서브쿼리를 사용하여 데이터 삭제하기)

- DELETE FROM EMP TEMP2

WHERE EMPNO IN (SELECT E.EMPNO
FROM EMP TEMP2 E,

SALGRADE S

WHERE E.SAL

BETWEEN

S.LOSAL AND S.HISAL

AND S.GRADE = 3

AND DEPTNO = 30);

(데이터 전체 삭제하기)

↳ 테이블에 있는 전체 데이터만 삭제하기

- DELETE FROM EMP TEMP2;

⑭ (18/11/17)

< 트랜잭션 제어와 세션 >

◦ 트랜잭션 이란? transaction

- 관계형 데이터베이스에서 하나의 작업 또는 일정하게 연관되어 있는 작업 수행을 위해 나눌 수 없는 최소 수행 단위
- 이러한 트랜잭션을 제어하는데 사용하는 명령어를 TCL 이라 한다.

Transaction Control Language.

ex) A계좌, B계좌 애제 활동시
A, B계좌 잔액이 동시에 처리되어야 함.

- (데이터에 입력·수정·삭제하기)

```
- INSERT INTO DEPT_TCL VALUES ('5', 'A', 'P');
  UPDATE DEPT_TCL SET LOC = 'C' WHERE
    DEPT_NO = 40;
  DELETE FROM DEPT_TCL WHERE DNAME = 'D';
  SELECT * FROM DEPT_TCL;
```

(트랜잭션을 완료하고 싶을 때)

- ROLLBACK;

(트랜잭션을 영원히 반영하고 싶을 때)

↑
모든 퀼드

- COMMIT; (영구히 반영)
ROLLBACK에도 안됨.

① 트랜잭션 진행 중 CREATE TABLE ~

② 서로운 트랜잭션 시작 INSERT, UPDATE,
(삭제) DELETE [ROLLBACK]

③ 서로운 트랜잭션 시작 INSERT, UPDATE,
(반영) DELETE [COMMIT]

< 세션과 읽기 일관성의 의미 >

◦ 세션이란? session

- 하나의 데이터베이스에 접속한 후 종료하기까지의 과정이 하나의 세션 ex) 고인 ~ 물고이 ~ 한 세션

- 어떤 활동을 위한 시간이나 기간.

◦ 트랜잭션은 데이터 조작 명령어가 모인 하나의 작업 단위를 뜻하며 세션 내부에는 하나 이상의 트랜잭션이 존재한다.

→ 세션 > 트랜잭션

◦ 읽기 일관성? read consistency

- 어떤 데이터 조작이 포함된 트랜잭션이 완료 (COMMIT, ROLLBACK) 되기 전까지 데이터를 직접 조작하는 세션의 다른 세션에서는 데이터 조작 전 상태의 내용이 일관적으로 조회·출력·검색하는 특성.

* ROLLBACK으로 명령어 수행이 취소될 경우에 대하여 변경 전 데이터를undo segment)에 따로 저장.

(15) (18/11/1)

<수정중인 데이터 접근을 막는 LOCK>

- LOCK이란?
 - 특정 세션에서 조작 중인 데이터는 트랜잭션이 완료(Commit, Rollback)되기 전까지 다른 세션에서 조작할 수 없는 상태
 - HAVING(경) : 특정 세션에서 데이터 조작이 완료될 때까지 다른 세션에서 해당 데이터 조작을 기다리는 현상.
 - 행 레벨 락 (row level lock)
 - : 데이터 조작 관련 SQL문을 어떤 방식으로 작성하는데 따라 테이블의 일부 데이터만 LOCK이 될 수도 있고 테이블 전체 데이터가 LOCK이 될 수 있다.

<객체를 생성, 변경, 삭제하는 데이터 정의>

• 데이터 정의어 (DDL: Data Definition Language)

: 데이터베이스 데이터를 보관하고 관리하기 위해 제공되는 여러 객체(object)의 생성·변경·삭제 관련 기능을 수행

↳ 데이터 조작어 (DML)와 달리 명령어를 수행하자마자 데이터베이스에 수행한 내용이 바로 반영되는 특성

즉, 데이터 정의어를 실행하면 자동으로 COMMIT되기 때문에... ROLLBACK으로 실행취소 불가 그래서 사용시 주의 필요.

- CREATE : 객체를 생성하는
- ALTER : 이미 생성된 객체를 변경하는
- DROP : 객체를 삭제하는

※ TIP: 원도우 유틸리티 같은 'FLASH BACK' 기능으로 보호하지 있다 (DROP으로 삭제된 테이블)

• 테이블을 생성하는 CREATE

* 생성 규칙

- 1) 테이블 이름은 문자로 시작 (숫자 시작 X)
- 2) " 30 byte 이하
- 3) 같은 테이블명 사용 불가
- 4) 테이블 이름은 영문/한글, 숫자, 특수문자 사용 가능
- 5) SQL 쿼리를 테이블 이름으로 사용 불가 (SELECT, FROM 등)

* 열 이름 생성 규칙

- 1) 문자로 시작
- 2) 30 byte 이하
- 3) 열 이름 중복 X (한 테이블 내)
- 4) 영문/한글/숫자/특수문자 사용 가능
- 5) SQL 쿼리를 사용하지

(자료형을 각각 정하여 새 테이블 생성)

- CREATE TABLE EMP_DDL (
EMPNO NUMBER(4),
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
MGR NUMBER(4),
HIREDATE DATE,
SAL NUMBER(7,2),
COMM NUMBER(7,2),
DEPTNO NUMBER(2));
DESC EMP_DDL;

(기존 테이블 열 구조의 데이터를 복사하여 새 테이블 생성)

- CREATE TABLE DEPT_DDL
AS SELECT * FROM DEPT;

(일부 데이터만)

- CREATE TABLE EMP_DDL_30
AS SELECT *
FROM EMP
WHERE DEPTNO = 30;

(열 구조만 복사)

- ~ WHERE 1 < > 1 ;

⑯ (181117)

〈테이블을 변경하는 ALTER〉

- ALTER [테이블에 새 열을 추가 또는 삭제
 └ 열의 자료형 또는 길이를 변경]
 - 테이블에 열 추가하는 ADD
 - ALTER TABLE EMP_ALTER
 ADD HP VARCHAR2(20);
 추가할 이름 선택형.
 - 열 이름을 변경하는 RENAME
 - ALTER TABLE EMP_ALTER
 RENAME COLUMN HP TO TEL;
 기존열 이름 신규 이름.
 - 열 자료형을 변경하는 MODIFY
 - ALTER TABLE EMP_ALTER
 MODIFY EMPNO NUMBER(5);
 열 이름 변경과하는 형식.
 - 특정 열을 삭제할 때 사용하는 DROP
 - ALTER TABLE EMP_ALTER
 DROP COLUMN TEL;
 삭제할 열 이름.

<테이블 이름 변경하는 RENAME>

<테이블의 데이터를 삭제하는 TRUNCATE>

- TRUNCATE TABLE EMP - RENAME;
↳ 대신 DELETE FROM 사용
하지만 차이점 TRUNCATE (DDL) → ROLLBACK 이뤄지 않음.
DELETE (DML)

〈테이블을 삭제하는 DROP〉

- DROP TABLE EMP_RENAME;

〈 객체 종류 〉

- o 객체
 - 테이블
 - 데이터 사전 (data dictionary)
 - 인덱스 (index)
 - 뷰 (view)
 - 시퀀스 (sequence)
 - 동의어 (synonym)

• 데이터 사진? (329 p)

- 데이터베이스 메모리·성능·사용자 권한·객체 등
오류를 데이터베이스 운영에 중요한 데이터 보관.
만약 이 데이터에 문제가 발생한다면
오류를 데이터베이스 사용이 불가

○ 사용 가능한 데이터 자료를 알고 싶다면

- SELECT * FROM DICT;
 - SELECT * FROM DICTIONARY;

◦ 유저 테이블 (USER_접두어)

- SELECT TABLE NAME
FROM USER_TABLES ;

o ALL_접두어

- SELECT OWNER, TABLE_NAME
FROM ALL_TABLES;
↳ 유저테이블과 달리 OWNER 열이 더 많음
테이블의 소유자 정보.

o DBA_ 접두어

- SELECT * FROM DBA_TABLES;

▶ AI로 가는 정부 알아보기

- ```
- SELECT * FROM DBA_USERS
 WHERE USERNAME = 'SCOTT'
```

(17) (18/11/17)

<더 빠른 접근을 위한 인덱스>

◦ 인덱스란?

: 오라클 데이터베이스에서 데이터 검색 성능의 향상을 위해 테이블 열에 사용하는 객체

◦ 인덱스 사용여부에 따라

데이터 검색 방식  
Table Full Scan  
Index Scan

◦ 인덱스 정보 알아보기

- SELECT \*  
FROM USER\_INDEXES;

◦ 인덱스 열별 정보 알아보기

- SELECT \*  
FROM USER\_IND\_COLUMNS;

↳ 인덱스는 사용자가 직접 칼럼에 적용 할 수 있지만 primary key or unique key는 자동으로.

◦ 인덱스 생성 (336 P)

- CREATE INDEX IDX\_EMP\_SAL  
ON EMP(SAL);

◦ 인덱스 삭제

- DROP INDEX 인덱스 이름;

<테이블처럼 사용하는 뷰>

◦ 뷰란?

흔히 가상 테이블(virtual table)로 불림.

하나 이상의 테이블을 조합하는 SELECT문을 저장한 객체.

\*사용 이유 [ 편의성 (특정로 )]  
보안성 (테이블 특권을 둘 때)

\* 시스템 계정 (계정번호) 접속하기.

명령 프롬프트에서

SQLPLUS SYSTEM/oracle

GRANT CREATE VIEW TO SCOTT;

권한 부여후에

토드에서 SCOTT 계정으로 접속하여 뷰 생성.

◦ 뷰 생성

- CREATE VIEW VW\_EMP20  
AS (SELECT EMPNO, ENAME, JOB, DEPTNO,  
FROM EMP  
WHERE DEPTNO = 20);

(생성 뷰 확인해 보기)

- SELECT \*  
FROM USER\_VIEWS;

◦ 뷰 삭제

- DROP VIEW VW\_EMP20;

↳ 뷰는 실제 데이터가 아닌 SELECT문으로 저장하므로  
뷰를 삭제해도 테이블이나 데이터가 삭제되는 것은 아니다.

(18) (18/11/17)

## 인라인 뷰를 사용한 TOP-N SQL

### inline view

↳ 일회성으로 만들어 사용하는 뷰

ex) SELECT 문에서 사용되는 서브쿼리,  
WITH 절에서 여러 이름을 정의해 두고  
사용하는 SELECT 문

① ROWNUM (조회된 행 순서대로 매겨진 일련번호)  
↳ 의사 열 (pseudo column)

- SELECT ROWNUM, E.\*  
FROM EMP E;

↳ ROWNUM 열은 EMP 테이블에  
존재하지는 않지만 ROWNUM  
열의 데이터가 숫자로 출력

② 서브쿼리 사용 인라인 뷰 (345P)

- SELECT ROWNUM, E.\*  
FROM (SELECT \*  
FROM EMP E  
ORDER BY SAL DESC) E;

③ WITH 절 사용 인라인 뷰

- WITH E AS (SELECT \*  
FROM EMP  
ORDER BY SAL DESC)  
SELECT ROWNUM, E.\*

FROM E;

## 인라인 뷰를 사용한 TOP-N 추출

### 서브쿼리 사용

- SELECT ROWNUM, E.\*  
FROM (SELECT \* FROM EMP E  
ORDER BY SAL DESC) E  
WHERE ROWNUM <= 3;

### WITH 절 사용

- WITH E AS (SELECT \* FROM EMP  
ORDER BY SAL DESC)  
SELECT ROWNUM, E.\*  
FROM E  
WHERE ROWNUM <= 3;

< 규칙에 따라 순번을 생성하는 서비스 >

### 서비스 ?

Sequence는 오라클 데이터베이스에서  
특정 규칙에 맞는 연속 숫자를 생성하는 객체

### 서비스 생성하기

- CREATE SEQUENCE SEQ\_DEPT\_SEQUENCE  
INCREMENT BY 10 → 일관성 유지하는지 의미  
START WITH 10 → 시작 번호 의미  
MAXVALUE 90  
MINVALUE 0  
NOCYCLE → MAX 도달하면 끝  
CACHE 2, 반대로 CYCLE 옵션은  
다시 처음부터 즐립

### 생성한 서비스 확인하기

- SELECT \*  
FROM USER\_SEQUENCES;

### 서비스 사용 (350P)

(서비스에서 생성한 순번을 사용한 INSERT를 실행하기)

- INSERT INTO DEPT\_SEQUENCE (DEPTNO,  
DNAME, LOC)  
VALUES (SEQ\_DEPT\_SEQUENCE.NEXTVAL,  
'DATABASE', 'SEOUL');

\* NEXTVAL은 다음번호를 생성 의미  
CURRVAL은 서비스에서 마지막으로 생성한 번호 반환

(가장 마지막으로 생성된 서비스 확인)

- SELECT SEQ\_DEPT\_SEQUENCE.CURRVAL  
FROM DUAL;

(서비스에서 생성한 순번을 반복 사용)

→ INSERT 를 계속 9번 반복하여  
실행하면 DEPT\_SEQUENCE 테이블에  
9개의 행이 생성됩니다.

10

20

30

⋮

90

예: MAXVALUE 값이 90  
(서비스 생성 시)

→ 한 번 더 실행하면 실패 일정

⑯ (18/11/7 ~ 18/11/8)

◦ 시퀀스 수정 (ALTER 명령어)

\* 참고로 START WITH 같은 변경 불가

- ALTER SEQUENCE SEQ\_DEPT\_SEQUENCE

INCREMENT BY 3;

MAX VALUE 99

CYCLE;

NOCYCLE 대신 CYCLE을 원 선택의지

- (수정 시퀀스 사용 실행)

- INSERT INTO DEPT\_SEQUENCE (DEPTNO,  
DNAME, LOC)

VALUES (SEQ\_DEPT\_SEQUENCE.NEXTVAL,  
'DATA BASE', 'SEOUL');

SELECT \* FROM DEPT\_SEQUENCE  
ORDER BY DEPTNO;

◦ 시퀀스 삭제 (DROP 명령어)

- DROP SEQUENCE SEQ\_DEPT\_SEQUENCE;

SELECT \* FROM USER\_SEQUENCES;

<공식 별칭을 지정하는 동의어>

◦ 동의어 ?

Synonym 는 테이블, 뷰, 시퀀스 등  
객체 이름 대신 사용할 수 있는 다른 이름을  
부여하는 객체

◦ 동의어 생성 (SYSTEM 계정 접속하여 SCOTT 계정)

- CREATE SYNONYM E FOR EMP;

- SELECT \* FROM E; 확인 가능.

◦ 동의어 삭제

- DROP SYNONYM E;

↳ 동의어로 E를 사용할 수 없지만  
기존 EMP 테이블은 그대로.

※ 동의어를 생성하고 사용하기 위해서는  
생성 과정을 봐야 한다.

(명령어 프롬프트 접속)

- SQLPLUS SYSTEM / oracle  
계정번호 비밀번호

- GRANT CREATE STANONYM TO SCOTT;  
SCOTT 계정

- GRANT CREATE PUBLIC STANONYM  
TO SCOTT;

동의어를 데이터베이스 내 모든 사용자가  
사용할 수 있도록 설정.  
생각할 경우 동의어를 생성한 사용자만  
사용 가능.

<제약 조건> Constraint

◦ 제약 조건 ?

: 테이블에 저장할 데이터를 제약하는 특수한 규칙  
제약조건을 설정한 열에는 그것에 맞지 않는  
데이터를 저장할 수 없습니다.

제약조건은 테이블의 특정 열에 지정한다.

◦ 제약 조건 종류

1) NOT NULL : 지정한 열에 NULL 허용 X  
(NULL을 제외한 중복 데이터 허용)

2) UNIQUE : 지정한 열에 유일한 값  
(즉 중복 안됨, NULL값은 중복 제외)

3) PRIMARY KEY : 지정한 열이 유일한 값  
NULL 허용 X  
테이블에 하나만 지정 가능

4) FOREIGN KEY : 다른 테이블의 열을 참조하여  
존재하는 값만 입력 가능

5) CHECK : 설정한 조건을 만족하는  
데이터만 입력 가능.

② (18/11/18)

- 데이터 무결성 (data integrity)
  - : 데이터베이스에 저장되는 데이터의 정확성과 일관성을 보장한다는 의미
  - ✓ 이를 위해 '제약 조건'이 중요한 역할
- 데이터 무결성의 종류
  - 1) 영역 무결성 (domain integrity)
    - : 열에 저장되는 값의 적정 여부를 확인
  - 2) 개체 무결성 (entity integrity)
    - : 테이블 데이터를 유일하게 식별할 수 있는 기본키는 반드시 값을 가지고 있어야 하며 NULL이 될 수 없고  
중복될 수도 없음을 규정
  - 3) 참조 무결성 (referential integrity)
    - : 참조 테이블의 외래키 값은 참조 테이블의 기본기로서 존재해야 하며 NULL이 가능

- NOT NULL
  - : 지정한 열에 NULL의 저장을 허용하지 않음.
- CREATE TABLE TABLE\_NOTNULL (  
    LOGIN-ID VARCHAR2(20) NOT NULL,  
    LOGIN-PWD VARCHAR2(20) NOT NULL,  
    TEL        VARCHAR2(20)  
);  
  
    ↓  
    INSERT 문, UPDATE 문으로 NULL을 넣어도  
    실행된다.

- 제약 조건 확인 방법.
  - SELECT OWNER, CONSTRAINT\_NAME,  
        CONSTRAINT\_TYPE, TABLE\_NAME  
    FROM USER\_CONSTRAINTS;
  - C: CHECK, NOT NULL
  - U: UNIQUE
  - P: PRIMARY KEY
  - R: FOREIGN KEY

◦ 제약 조건 이름 직접 지정

```
- CREATE TABLE TABLE_NOTNULL_2 (
 LOGIN-ID VARCHAR2(20) CONSTRAINT
 TBLNN2-LGNID_NN NOT NULL,
 LOGIN-PWD VARCHAR2(20) CONSTRAINT
 TBLNN2-LGNPW_NN NOT NULL,
 TEL VARCHAR2(20)
);
```

- 이미 생성한 테이블에 제약 조건 지정  
(TEL 열에 NOT NULL 제약 조건 추가하기)

```
- ALTER TABLE TABLE_NOTNULL
MODIFY (TEL NOT NULL)
```

↓  
이미 TEL 열에 NULL 값이 존재한다면  
예외 발생 (제약 조건 지정 불가)

↓  
(TEL 열 데이터 수정하기)

```
- UPDATE TABLE_NOTNULL
SET TEL = '010-1234-5678'
WHERE LOGIN-ID = 'TEST_ID-01'
```

↓  
다시 제약 조건 지정 → 해변 유희.

- 생성한 테이블에 제약 조건 이름 직접 지정해서 추가

```
- ALTER TABLE TABLE_NOTNULL_2
MODIFY (TEL CONSTRAINT TBLNN-TEL-NN NOT NULL)
```

```
SELECT OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE,
 TABLE_NAME
FROM USER_CONSTRAINTS;
```

- 열 → 조 확인하기

```
- DESC TABLE_NOTNULL_2;
```

- 이미 생성된 제약 조건 이름 변경하기

```
- ALTER TABLE TABLE_NOTNULL_2
RENAME CONSTRAINT TBLNN-TEL-NN TO TBLNN2-TEL-NN;
```

- 제약 조건 삭제하기

```
- ALTER TABLE TABLE_NOTNULL_2
DROP CONSTRAINT TBLNN2-TEL-NN;
```

② (18/11/18)

## ◦ UNIQUE

: 지정한 열에 데이터의 중복을 허용하지 않음.  
(단, NULL 같은 중복 허용)

- CREATE TABLE TABLE\_UNIQUE (  
    LOGIN\_ID VARCHAR2(20) UNIQUE,  
    LOGIN\_PWD VARCHAR2(20) NOT NULL,  
    TEL      VARCHAR2(20)  
)

DESC TABLE\_UNIQUE

## ◦ 제약조건 확인

- SELECT OWNER, CONSTRAINT\_NAME,  
    CONSTRAINT\_TYPE, TABLE\_NAME  
    FROM USER\_CONSTRAINTS  
    WHERE TABLE\_NAME = 'TABLE\_UNIQUE';

## ◦ 중복을 허락하지 않는 UNIQUE

(TABLE\_UNIQUE 테이블에 데이터 입력하기)  
- INSERT INTO TABLE\_UNIQUE (LOGIN\_ID,  
    LOGIN\_PWD, TEL)  
VALUES ('TEST-ID-01', 'PWD01', '010-1234-5678');

↓  
열에 중복되는 데이터 넣기

→  
↓

예외발생 (중복금지)

## ◦ UNIQUE 제약 조건이 지정된 열에 NULL 값 입력하기

- INSERT INTO TABLE\_UNIQUE (LOGIN\_ID,  
    LOGIN\_PWD, TEL)  
VALUES (NULL, 'PWD01', '010-2345-6789');

## ◦ TABLE-UNIQUE 테이블 데이터 수정하기.

- UPDATE TABLE\_UNIQUE  
    SET LOGIN\_ID='TEST-ID-01'  
    WHERE LOGIN\_ID IS NULL;  
↓  
예외발생 (기존에 TEST-ID-01으로서 중복)

## ◦ 테이블 생성 UNIQUE 제약 조건 설정

- CREATE TABLE TABLE\_UNIQUE2 (

    LOGIN\_ID VARCHAR2(20) CONSTRAINT  
        TBLUNQ2-LEID-UNQ UNIQUE,  
    LOGIN\_PWD VARCHAR2(20) CONSTRAINT  
        TBLUNQ2-LEPWD-UN NOT NULL,  
    TEL      VARCHAR2(20)  
)

## ◦ 생성한 UNIQUE 제약 조건 확인

- SELECT OWNER, CONSTRAINT\_NAME,  
    CONSTRAINT\_TYPE, TABLE\_NAME  
    FROM USER\_CONSTRAINTS  
    WHERE TABLE\_NAME LIKE 'TABLE\_UNIQUE%';

## ◦ 이미 생성한 테이블 열에 UNIQUE 제약 조건 추가

- ALTER TABLE TABLE\_UNIQUE  
MODIFY (TEL UNIQUE);

↓

예외발생 (TEL 열에 이미 중복 허용되었음)

(TEL 열 값을 NULL 값 변경)

- UPDATE TABLE\_UNIQUE  
SET TEL = NULL;

↓

→ 다시 제약조건 추가 하면 안됨.

## ◦ 이미 생성한 테이블 UNIQUE 제약 조건 이름 지정

- ALTER TABLE TABLE\_UNIQUE2  
MODIFY (TEL CONSTRAINT TBLUNQ2-TEL-UNQ UNIQUE);

## ◦ 이미 만들어져 있는 UNIQUE 이름 수정.

- ALTER TABLE TABLE\_UNIQUE2  
RENAME CONSTRAINT TBLUNQ2-TEL-UNQ TO 바꿀이름.

## ◦ 제약조건 삭제

- ALTER TABLE TABLE\_UNIQUE2  
DROP CONSTRAINT TBLUNQ2-TEL-UNQ;

(22) (18/11/18)

## o PRIMARY KEY

- UNIQUE와 NOT NULL 제약 조건 특성을 모두 가지는 제약 조건  
즉, 중복 허용 X, NULL 허용 X

## o 테이블 생성시 PRIMARY KEY 설정.

- CREATE TABLE TABLE\_PK(

DEPTN\_ID VARCHAR2(20) PRIMARY KEY,  
DEPTN\_PWD VARCHAR2(20) NOT NULL,  
TELL VARCHAR2(20)

)!

## o 생성한 Primary key 확인

- SELECT OWNER, CONSTRAINT\_NAME,  
CONSTRAINT\_TYPE, TABLE\_NAME  
FROM USER\_CONSTRAINTS  
WHERE TABLE\_NAME LIKE 'TABLE\_PK%';

## o 생성한 Primary key를 통해 자동 생성 INDEX

- SELECT INDEX\_NAME, TABLE\_OWNER  
TABLE\_NAME  
FROM USER\_INDEXES  
WHERE TABLE\_NAME LIKE 'TABLE\_PK%';

## o 다른 방식으로 제약 조건 이름 지정, 제약 조건 지정.

- CREATE TABLE TABLE\_NAME(  
영이름1 VARCHAR2(20),  
영이름2 VARCHAR2(20),  
영이름3 VARCHAR2(20),  
PRIMARY KEY (영이름1),  
CONSTRAINT CONSTRAINT\_NAME UNIQUE(  
제약조건 이름),  
영이름2  
);

)!

## o FOREIGN KEY (외래키, 외부키)

- 서로 다른 테이블 간 관계를 정의하는 데 사용하는 제약 조건

ex) EMP 테이블의 DEPTNO 열은

DEPT 테이블의 primary key인 DEPTNO 열 값 (10, 20, 30, 40)과 NULL 이외의 다른 저장이 불가

그래서 아래와 같이 EMP의 DEPTNO 열에 '10'을 넣으면 여러 번생

(이유: DEPT의 DEPTNO 열에 'E.' 있기 때문)

- INSERT INTO EMP(EMPNO, ENAME, JOB,  
MGR, HIREDATE, SAL,  
COMM, DEPTNO)  
VALUES (9999, '홍길동', 'CLERK', '788',  
TO\_DATE('2017/04/30', 'YYYY/MM/DD'),  
1200, NULL, 50);

↓

여러번생.

## o FOREIGN KEY 지정 방법.

(DEP\_FK 테이블 생성)

- CREATE TABLE DEPT\_FK(  
DEPTNO NUMBER(2) CONSTRAINT — PRIMARY KEY,  
DNAME VARCHAR2(44),  
LOC VARCHAR2(13) )!

(EMP-FK 테이블 생성)

- CREATE TABLE EMP\_FK(  
EMPNO NUMBER(4) CONSTRAINT — 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 —  
REFERENCES DEPT\_FK(DEPTNO)

)!

DESC EMP\_FK;

\*주의점: 테이블을 만들고 나서 DEPT\_FK 테이블에는  
데이터가 아직 없는 상태이기 때문에 EMP\_FK 테이블



② (18/11/19)

## < 사용자, 권한, 틀 관리 >

### ◦ 사용자 (USER)

: 데이터베이스에 접속하여 데이터를 관리하는 계정

### ◦ 데이터베이스 스키마 (schema)

: 오라클 데이터베이스에 접속한 사용자와 연결된 객체

ex) SCOTT 계정 → 사용자

SCOTT가 생성한 테이블·뷰·제약조건  
인덱스·시퀀스·동의어 등

데이터베이스에서 SCOTT 계정으로 만든 모든 객체는 SCOTT의 스키마가 된다.

### ◦ 사용자 생성 (시스템계정으로 접속 후)

- CREATE USER ORCLSTUDY 계정명  
IDENTIFIED BY ORACLE!  
비밀번호

### ◦ 접속권한부여 (시스템계정으로 접속 후)

- GRANT CREATE SESSION TO ORCLSTUDY;  
권한부여  
↳ 데이터베이스 연결을 위한 권한

### ◦ NEW 사용자 계정 (ORCLSTUDY)에 접속하기

- CONN ORCLSTUDY/ORACLE  
접속도 사용자 계정 비밀번호

### ◦ 사용자 정보 조회 (데이터사전 사용)

- SELECT \* FROM ALL-USERS  
WHERE USERNAME = 'ORCLSTUDY';

- SELECT \* FROM DBA-USERS  
WHERE USERNAME = 'ORCLSTUDY';

- SELECT \* FROM DBA-OBJECTS  
WHERE OWNER = 'ORCLSTUDY';

### ◦ 사용자 정보(패스워드) 변경하기

- ALTER USER ORCLSTUDY  
IDENTIFIED BY ORCL;

### ◦ 오라클 사용자 삭제 (마을로 접속시, 삭제시 예외발생)

- DROP USER ORCLSTUDY;

### ◦ 오라클 사용자와 객체(스키마) 모두 삭제

- DROP USER ORCLSTUDY CASCADE;

## < 권한 관리 >

접속 사용자에 따라 접근할 수 있는 데이터 영역과 권한을 지정해 줄 수 있다.

### ◦ 시스템 권한 (system privilege)

### ◦ 객체 권한 (object privilege)

#### ◦ 시스템 권한 (system privilege)

: 사용자 생성, 정보수정/삭제, DB접근, 객체생성/관리

#### ◦ 시스템 권한 부여

- GRANT CREATE SESSION TO ORCLSTUDY;  
접속권한

- GRANT RESOURCE, CREATE TABLE  
TO ORCLSTUDY;

↓  
테이블 생성 CREATE TABLE();

↓  
열에 행 데이터 생성 INSERT INTO  
VALUES();

#### ◦ % RESOURCE (%)

만약 지정하지 않으면, 테이블 생성 권한이  
부여되더라도 CREATE or INSERT 등  
제대로 작동하지 않을 수 있다.

(즉, 여러 권한을 하나의 이름으로 묶어  
접근부여하고자 작업을 간편하게 하려고 한  
'role'이다)

#### ◦ 시스템 권한 취소

- REVOKE RESOURCE, CREATE TABLE  
FROM ORCLSTUDY;

#### ◦ 객체 권한 (object privilege)

: 테이블·인덱스·뷰·시퀀스 등의 권한

#### ◦ 객체 권한 부여 (, 한번에 지정 가능)

- GRANT SELECT ON TEMP TO ORCLSTUDY;  
↳ 템파일, 객체이름

- " " INSERT " " " " "

#### ◦ 객체 권한 취소

- REVOKE SELECT, INSERT ON TEMP  
FROM ORCLSTUDY;

25 (18/11/9)

## <롤 관리>

- : role은 여러 종류의 권한을 묶어 놓은 그룹.  
↳ role을 사용하여 권한을 부여/제거가능 편리함.

사전 정의된 틀 (predefined roles)

사용자 정의 틀 (user roles)

## o predefined roles (사전 정의된 틀)

### 1) CONNECT 틀

ALTER SESSION, CREATE CLUSTER  
CREATE DATABASE LINK, CREATE SEQUENCE,  
CREATE SESSION, CREATE SYNONYM,  
CREATE TABLE, CREATE VIEW

### 2) RESOURCE 틀

CREATE TRIGGER, CREATE SEQUENCE,  
CREATE TYPE, CREATE PROCEDURE,  
CREATE CLUSTER, CREATE OPERATOR,  
CREATE INDEXTYPE, CREATE TABLE

### 3) DBA 틀

: 데이터베이스를 관리하는 대부분의 기능

## o user roles (사용자 정의 틀)

### 1) 틀 만들기

- CREATE ROLE ROLESTUDY;  
    <sub>롤이름</sub>

- GRANT CONNECT, RESOURCE, CREATE VIEW,  
    CREATE SYNONYM TO ROLESTUDY;

### 2) 사용자 (ORCLSTUDY)에게 권한 부여

- GRANT ROLESTUDY TO ORCLSTUDY;  
    <sub>new role</sub>                <sub>사용자</sub>

### 3) 틀 취소

- REVOKE ROLESTUDY FROM ORCLSTUDY;

### 4) 틀 삭제

- DROP ROLE ROLESTUDY;

## <PL/SQL 기초>

: 오라클에서 제공하는 프로그래밍 언어

## <PL/SQL 구조>

### o 블록 (block)

: 명령어를 모아둔 PL/SQL 프로그램의 기본 단위

1) DECLARE (선언부) : 실행에 사용될 변수, 상수, 커서 등 선언

2) BEGIN (실행부) : 조건문, 반복문, SELECT, DML - 함수 정의

3) EXCEPTION (예외 처리부) : PL/SQL 실행 도중 발생하는 오류 (예외 상황)를 해결하는 문장 기호

### o Hello PL/SQL 출력하기 (예제)

- SET SERVEROUTPUT ON; — 실행결과 하면 출력  
BEGIN  
    DBMS\_OUTPUT.PUT\_LINE('Hello, PL/SQL!');  
END;  
/

## ※ 주의 사항

1) DECLARE, BEGIN, EXCEPTION에는 ';' 사용금지

2) 블록 각 부분 실행 문장 끝에는 세미콜론 ';' 사용

3) 한 줄 주석 (--) , 여러 줄 주석 (/ \* ~ \* /)

4) PL/SQL문을 마치고 실행을 위해  
마지막에 ;(semicolon)을 사용합니다.

② (18/11/19)

## o 변수와 상수

(기본 변수 선언과 사용)

### - DECLARE

```
 V-EMPNO NUMBER(4) := 7788;
 V-ENAME VARCHAR2(10);
```

값 할당.

BEGIN

V-ENAME := 'SCOTT';

↳ 문자열 SCOTT 저장

```
 DBMS_OUTPUT.PUT_LINE('V-EMPNO:' || V-EMPNO);
 DBMS_OUTPUT.PUT_LINE('V-ENAME:' || V-ENAME);
```

END;

/

(결과하면)

V-EMPNO: 7788

V-ENAME: SCOTT

(상수 정의하기)

### - DECLARE

V-TAX CONSTANT NUMBER(1) := 3;

BEGIN

```
 DBMS_OUTPUT.PUT_LINE('V-TEX:' || V-TAX);
```

END;

/

↓

(결과하면) V-TEX: 3

(변수의 기본값 지정하기)

### - DECLARE

V-DEPTNO NUMBER(2) DEFAULT 10;

BEGIN

```
 DBMS_OUTPUT.PUT_LINE('V-DEPTNO:' || V-DEPTNO);
```

END;

/

↓

(결과하면) V-DEPTNO: 10

(변수에 NULL 값 저장 막기)

### - DECLARE

V-DEPTNO NUMBER(2) NOT NULL := 10;

BEGIN

```
 DBMS_OUTPUT.PUT_LINE('V-DEPTNO:' || V-DEPTNO);
```

END;

/

↓  
(결과하면) V-DEPTNO: 10

(변수 NOT NULL 및 기본값 동시에)

V-DEPTNO NUMBER(2) NOT NULL DEFAULT 10;

## o 변수의 자료형

Scalar (스칼라)

Composite (복합)

reference (참조)

Large Object (LOB)

### (스칼라형)

: 숫자, 문자열, 날짜의 기본자료형

### (참조형)

: 특정 테이블의 열 아래의 주소를 참조하는 자료형

ex) %TYPE, %ROWTYPE

열 참조, 행 참조

### DECLARE

V-DEPTNO DEPT.DEPTNO%TYPE := 50;

BEGIN

↑ 열참고  
앞의 같은 자료형, → 기호를 의미

```
 DBMS_OUTPUT.PUT_LINE('V-DEPTNO:' || V-DEPTNO);
```

END;

/

(27) (18/11/9)

### <조건 제어문>

DECLARE

V-DEPT\_ROW DEPT% ROWTYPE ;

BEGIN

SELECT DEPTNO, DNAME, LOC  
INTO V-DEPT\_ROW  
FROM DEPT  
WHERE DEPTNO = 40;

DBMS\_OUTPUT.PUT\_LINE('DEPTNO:' || V-DEPT\_ROW.DEPTNO);  
" " " " ('DNAME:' || V-DEPT\_ROW.DNAME);  
" " " " ('LOC:' || V-DEPT\_ROW.LOC);

END;

/

↓ (결과)

DEPTNO = 40

DNAME = OPERATIONS

LOC = BOSTON

(부수형)

— 칼럼 : TABLE (테이블의 열과 유사)

— 레코드 : RECORD (테이블의 행과 유사)

(LOB형) Large Object.

: 대용량의 텍스트, 이미지, 동영상, 사운드 데이터를  
저장하기 위한 자료형.

CLOB, BLOB 등 있다)

o IF 조건문

— IF-THEN  
— IF-THEN-ELSE  
— IF-THEN-ELSIF

o IF-THEN

DECLARE

V\_NUMBER NUMBER := 13;

BEGIN

IF MOD(V\_NUMBER, 2) = 1 THEN  
DBMS\_OUTPUT.PUT\_LINE('V\_NUMBER는 홀수');

END IF;

END;

/

↓ (결과)

V\_NUMBER는 홀수.  
13

o IF-THEN-ELSE

DECLARE

V\_NUMBER NUMBER := 14;

BEGIN

IF MOD(V\_NUMBER, 2) = 1 THEN  
DBMS\_OUTPUT.PUT\_LINE('V\_NUMBER는 홀수');

ELSE

DBMS\_OUTPUT.PUT\_LINE('V\_NUMBER는 짝수');

END IF;

END;

/

↓  
V\_NUMBER는 짝수입니다.

②8) (18/11/19)

## ◦ IF - THEN - ELSEIF

DECLARE

V\_SCORE NUMBER := 87;

BEGIN

IF V\_SCORE >= 90 THEN

DBMS\_OUTPUT.PUT\_LINE('A학점');

ELSIF V\_SCORE >= 80 THEN

DBMS\_OUTPUT.PUT\_LINE('B학점');

ELSIF V\_SCORE >= 70 THEN

DBMS\_OUTPUT.PUT\_LINE('C학점');

ELSIF " >= 60 "

" " " ("D학점");

ELSE

DBMS\_OUTPUT.PUT\_LINE('F학점');

END IF;

END;

/

↓

B학점.

◦ CASE 조건문

단순 CASE문 = 비교문 명령 O  
간접 CASE문 = 비교문 명시 X

(단순 CASE)

DECLARE

V\_SCORE NUMBER := 87;

BEGIN

CASE TRUNC(V\_SCORE/10)

WHEN 10 THEN DBMS\_OUTPUT.PUT\_LINE('A');

WHEN 9 THEN " " " ("B");

WHEN 8 THEN " " " ("C");

WHEN 7 THEN " " " ("D");

ELSE DBMS\_OUTPUT.PUT\_LINE('F');

END CASE;

END;

/

↓

(간접 CASE)

DECLARE

V\_SCORE NUMBER := 87;

BEGIN

CASE

WHEN V\_SCORE >= 90

THEN DBMS\_OUTPUT.PUT\_LINE('A');

WHEN V\_SCORE >= 80

THEN .. .. ("B");

WHEN V\_SCORE >= 70

THEN .. .. ("C");

WHEN V\_SCORE >= 60

THEN .. .. ("D");

ELSE DBMS\_OUTPUT.PUT\_LINE('F');

END CASE;

END

/

↓

B

< 반복 제어문 >

기본 LOOP 기본반복문

WHILE LOOP 특정 조건에 의해 반복

FOR LOOP 반복 횟수를 정하여 반복

Cursor FOR LOOP

가져온 할당한 반복

※ 반복 종료 명령어

EXIT 반복종료

EXIT-WHEN 종료 조건

CONTINUE 건너뛰기

CONTINUE-WHEN

조건에 주고 건너뛰기

29 (18/120)

## o 기본 LOOP

DECLARE

V\_NUM NUMBER := 0;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE('현재 V\_NUM: ' || V\_NUM);  
V\_NUM := V\_NUM + 1;

EXIT WHEN V\_NUM > 4;

END LOOP;

END;

/

DECLARE

V\_NUM NUMBER := 0;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE('현재 V\_NUM: ' || V\_NUM);

V\_NUM := V\_NUM + 1;

IF V\_NUM > 4 THEN

EXIT;

END IF;

END LOOP;

END;

/

## o WHILE LOOP

DECLARE:

V\_NUM NUMBER := 0;

BEGIN

WHILE V\_NUM < 4 LOOP

DBMS\_OUTPUT.PUT\_LINE('현재 V\_NUM: ' || V\_NUM);

V\_NUM := V\_NUM + 1;

END LOOP;

END;

/

o FOR LOOP (반복 횟수를 지정할 수 있다)

BEGIN

FOR i IN 0..4 LOOP

<sub>시작값</sub>   <sub>종료값</sub>

DBMS\_OUTPUT.PUT\_LINE('현재 i의 값: ' || i);

END LOOP;

END;

/

0  
1  
2  
3  
4

(역순으로 반복하고 싶다면)

BEGIN

FOR i IN REVERSE 0..4 LOOP

DBMS\_OUTPUT.PUT\_LINE('현재 i의 값: ' || i);

END LOOP;

END;

/

4  
3  
2  
1  
0

o CONTINUE문 & CONTINUE-WHEN문

(오라클 11g 버전 부터 사용) 조건문 컨티뉴하기,

BEGIN

FOR i IN 0..4 LOOP

CONTINUE WHEN MOD(i, 2) = 1;

DBMS\_OUTPUT.PUT\_LINE('현재 i의 값: ' || i);

END LOOP;

END;

/

0  
1  
2  
3  
4

설명: i=1일때 true 이므로 계속해서  
(CONTINUE)

i=2일때 false이고 DBMS\_3가지

없는다.

i=2일때 false가되면  
그다음 DBMS로 넘나간다.

③ (18/11/20)

## < 레코드와 컬렉션 >

• 레코드 ? record

: 자료형이 각기 다른 데이터를 하나의 별개에 저장하는 데 사용합니다.

DECLARE *→ 레코드 이름*

```
TYPE REC_DEPT IS RECORD (
 deptno NUMBER(2) NOT NULL := 99,
 dname DEPT.DNAME%TYPE,
 loc DEPT.LOC%TYPE
);
```

dept\_rec REC\_DEPT; *변수 선언*  
dept\_rec *별자리* 레코드 이름 *여러개의 변수 선언 가능*.

BEGIN

```
dept_rec.deptno := 99;
dept_rec.dname := 'DATABASE';
dept_rec.loc := 'SEOUL';
```

```
DBMS_OUTPUT.PUT_LINE('DEPTNO:' || dept_rec.deptno);
DBMS_OUTPUT.PUT_LINE('DNAME:' || dept_rec.dname);
DBMS_OUTPUT.PUT_LINE('LOC:' || dept_rec.loc);
```

END;

/

### (레코드를 사용한 INSERT)

```
CREATE TABLE DEPT_RECORD
AS SELECT * FROM DEPT;
```

DECLARE

```
TYPE REC_DEPT IS RECORD (
 deptno NUMBER(2) NOT NULL := 99,
 dname DEPT.DNAME%TYPE,
 loc DEPT.LOC%TYPE
);
```

dept\_rec REC\_DEPT;

↓ 개수

BEGIN

```
dept_rec.deptno := 99;
dept_rec.dname := 'DATABASE';
dept_rec.loc := 'SEOUL';
```

```
INSERT INTO DEPT_RECORD
VALUES dept_rec;
```

END;

/

### (레코드를 사용한 UPDATE)

DECLARE

```
TYPE REC_DEPT IS RECORD (
 deptno NUMBER(2) NOT NULL := 99,
 dname DEPT.DNAME%TYPE,
 loc DEPT.LOC%TYPE
);
```

dept\_rec REC\_DEPT;

BEGIN

```
dept_rec.deptno := 50;
dept_rec.dname := 'DB';
dept_rec.loc := 'SEOUL';
```

UPDATE DEPT\_RECORD

```
SET ROW = dept_rec
WHERE DEPTNO = 99;
```

END;

/

### (레코드를 포함하는 레코드)

DECLARE

```
TYPE REC_DEPT IS RECORD (
 depto DEPT.DEPTO%TYPE,
 dname DEPT.DNAME%TYPE,
 loc DEPT.LOC%TYPE
);
```

↓ 개수

③ (18/120)

↓이어서

```

TYPE REC_EMP IS RECORD (
 empno EMP.EMPNO%TYPE,
 ename EMP.ENAME%TYPE,
 dinfo REC_DEPT
);
emp_rec REC_EMP;

```

BEGIN

```

SELECT E.EMPNO, E.ENAME,
 D.DEPTNO, D.DNAME, D.LOC
 INTO emp_rec.empno,
 emp_rec.ename,
 emp_rec.dinfo.deptno,
 emp_rec.dinfo.dname,
 emp_rec.dinfo.loc

```

FROM EMP E, DEPT D

WHERE E.DEPTNO = D.DEPTNO  
AND E.DEPTNO = 7788;

```

DBMS_OUTPUT.PUT_LINE('EMPNO : '|| emp_rec.empno);
DBMS_OUTPUT.PUT_LINE('ENAME : '|| emp_rec.ename);
DBMS_OUTPUT.PUT_LINE('DEPTNO : '|| emp_rec.dinfo.deptno);
DBMS_OUTPUT.PUT_LINE('DNAME : '|| emp_rec.dinfo.dname);
DBMS_OUTPUT.PUT_LINE('LOC : '|| emp_rec.dinfo.loc);

```

END;

↓

```

EMPNO : 7788
ENAME : SCOTT
DEPTNO : 20
DNAME : RESEARCH
LOC : DALLAS

```

### ○ 결핵선

: 특정 자료형의 데이터를 여러 개 저장하는  
복합 자료형입니다.

\* 결핵선 종류

- 연관 배열 (associative array)
- 중첩 테이블 (nested table)
- VARRAY (variable-size array)

### ○ 연관 배열 (가장 사용 빈도 높음)

: 인덱스하고 볼리는 key, value 으로 구성

DECLARE

    연관 배열 이름

자료형  
연관 배열

    TYPE ITAB\_EX IS TABLE OF VARCHAR(20)  
    INDEX BY PLS\_INTEGER;  
    key 사용할

text\_arr ITAB\_EX;      인덱스의 자료형

BEGIN

```

text_arr(1) := '1st data';
text_arr(2) := '2nd data';
text_arr(3) := '3rd data';
text_arr(4) := '4th data';

```

```

DBMS_~ ('text_arr(1)'|| text_arr(1));
DBMS_~ ('text_arr(2)'|| text_arr(2));
DBMS_~ ('text_arr(3)'|| text_arr(3));
DBMS_~ ('text_arr(4)'|| text_arr(4));

```

END;

↓

```

text_arr(1) : 1st data
text_arr(2) : 2nd data
text_arr(3) : 3rd data
text_arr(4) : 4th data

```

③ (18/120)

(레코드를 활용한 연관 배열)

DECLARE

```
TYPE REC_DEPT IS RECORD (
 deptno DEPT. DEPTNO%TYPE,
 dname DEPT. DNAME%TYPE
);
```

```
TYPE ITAB_DEPT IS TABLE OF REC_DEPT
INDEX BY PLS_INTEGER;
dept_arr ITAB_DEPT;
idx PLS_INTEGER := 0;
```

BEGIN

```
FOR i IN (SELECT DEPTNO, DNAME
 FROM DEPT) LOOP
 idx := idx + 1;
 dept_arr(idx).deptno := i. DEPTNO;
 dept_arr(idx).dname := i. DNAME;
```

```
DBMS_OUTPUT.PUT_LINE(
 dept_arr(idx).deptno || ':' ||
 dept_arr(idx).dname);
END LOOP;
/
10 : ACCOUNTING
20 : RESEARCH
30 : SALES
40 : OPERATIONS.
```

% BINARY\_INTEGER  
PLS\_INTEGER ] 정수 자료형.

C %ROWTYPE으로 연관 배열 자료형 지정

DECLARE

```
TYPE ITAB_DEPT IS TABLE OF DEPT%ROWTYPE
INDEX BY PLS_INTEGER;
dept_arr ITAB_DEPT;
idx PLS_INTEGER := 0;
```

BEGIN

```
FOR i IN (SELECT * FROM DEPT) LOOP
 idx := idx + 1;
 dept_arr(idx).deptno := i. DEPTNO;
 dept_arr(idx).dname := i. DNAME;
 dept_arr(idx).loc := i. LOC;
```

DBMS\_OUTPUT.PUT\_LINE(

```
 dept_arr(idx).deptno || ':' ||
 dept_arr(idx).dname || ':' ||
 dept_arr(idx).loc);
```

END LOOP;

END;

↓  
10: ACCOUNTING : NEW YORK  
20: RESEARCH : DALLAS  
30: SALES : CHICAGO  
40: OPERATIONS : BOSTON

④ 컬렉션 메서드 (Able P) 상세내용은 책에서...

: 오라클에서는 컬렉션 사용상의 편의를 위해  
몇 가지 기본 프로그램을 제공하고 있는데  
이를 컬렉션 메서드라 한다.

(33) (18/11/23)

## <커서와 예외 처리>

반생 오류처리

### ○ 커서 Cursor

- : SELECT문 또는 데이터 조회와 같은 SQL문을 실행했을 때 해당 SQL문을 처리하는 정보를 저장한 메모리 공간을 뜻함
- \* 메모리 공간은 Private SQL Area라고 부르며 정확히 표현하자면 커서는 이 메모리의 포인터를 말합니다.

커서  
 명시적 (explicit) 커서  
 익시적 (implicit) 커서

### ○ SELECT INTO 방식

- 조회되는 데이터가 단 하나의 행 일 때 사용 가능한 방식
- 하지만 커서는 결과 행이 하나이든 여러개이든 상관없이 사용할 수 있습니다.

DECLARE

V\_DEPT\_ROW DEPT%ROWTYPE;

BEGIN

SELECT DEPTNO, DNAME, LOC <sup>개수와 자료형이 일치해야 한다</sup>  
INTO V\_DEPT\_ROW <sup>But 커서는 상관 없다.</sup>  
 FROM DEPT

WHERE DEPTNO = 40;

DBMS\_OUTPUT.PUT\_LINE('DEPTNO : ' ||  
 V\_DEPT\_ROW.DEPTNO);

DBMS\_OUTPUT.PUT\_LINE('DNAME : ' ||  
 V\_DEPT\_ROW.DNAME);

DBMS\_OUTPUT.PUT\_LINE('LOC : ' ||  
 V\_DEPT\_ROW.LOC);

END;

/

○ 명시적 (explicit) → 커서  
 (작성 방법)

DECLARE

CURSOR 커서 이름 IS SQL문; 커서 선언

BEGIN

OPEN 커서 이름; 커서 열기

FETCH 커서 이름 INTO 변수; 커서로부터 읽어온 데이터

CLOSE 커서 이름; 커서 닫기 사용

END;

(단일행 데이터를 저장하는 커서 사용하기)

DECLARE

V\_DEPT\_ROW DEPT%ROWTYPE; 커서 데이터를 입력할 변수 선언

CURSOR c1 IS -- 명시적 커서 선언

SELECT DEPTNO, DNAME, LOC

FROM DEPT

WHERE DEPTNO = 40;

BEGIN

OPEN c1; 커서 열기

= FETCH c1 INTO V\_DEPT\_ROW; 커서로부터 읽어온 데이터 사용

DBMS\_OUTPUT.PUT\_LINE('DEPTNO : ' ||

V\_DEPT\_ROW.DEPTNO);

DBMS\_OUTPUT.PUT\_LINE('DNAME : ' ||

V\_DEPT\_ROW.DNAME);

DBMS\_OUTPUT.PUT\_LINE('LOC : ' ||

V\_DEPT\_ROW.LOC);

CLOSE c1; 커서 닫기

END;

/

DEPTNO : 40

DNAME : OPERATIONS

LOC : BOSTON

(34) (18/11/23)

(여러행이 조회되는 경우 사용하는 LOOP문)

DECLARE

```
V_DEPT_ROW DEPT%ROWTYPE;
CURSOR C1 IS
SELECT DEPTNO, DNAME, LOC
FROM DEPT;
```

BEGIN

OPEN C1;

LOOP

```
FETCH C1 INTO V_DEPT_ROW;
EXIT WHEN C1%NOTFOUND;
-- 커서의 모든 행을 읽어오기 위해 속성 지정
DBMS_OUTPUT.PUT_LINE('DEPTNO:' ||
V_DEPT_ROW.DEPTNO ||
' , DNAME:' || V_DEPT_ROW.DNAME
|| ' , LOC : ' || V_DEPT_ROW.LOC);
```

END LOOP;

CLOSE C1;

END;

/

↓  
(실행 결과)

DEPTO: 10, DNAME: ACCOUNTING, LOC: NEW YORK  
" 20, " : RESEARCH, " : DALLAS  
" 30, " : SALES, " : CHICAGO  
" 40, " : OPERATIONS, " : BOSTON

\*

커서 이름 %NOTFOUND : 추출된 행이 있으면 false, 없으면 true

커서 이름 %FOUND : 추출된 행이 있으면 true, 없으면 false

커서 이름 %ROWCOUNT : 현재까지 추출된 행수 반환

커서 이름 %ISOPEN : 커서가 열려(open) 있으면 true  
닫혀(close) 있으면 false 반환

(FOR LOOP문을 활용하여 커서 사용하기)

↳ 여러 행이 조회되는 경우.

<작성법>

```
FOR 주프 인덱스 이름 IN 커서 이름 LOOP
 결과 행별로 반복 수행할 작업!
END LOOP;
```

- 주프 인덱스 (Loop Index)는 커서에 저장된 각 행이 저장되는 번수를 뜻하면 '.'을 통해 행의 각 필드에 접근할 수 있다.

- FOR LOOP문을 사용하면 OPEN, FETCH, CLOSE를 작성하지 않습니다.

DECLARE

```
CURSOR C1 IS
SELECT DEPTNO, DNAME, LOC
FROM DEPT;
```

BEGIN

FOR C1\_rec IN C1 LOOP (작동open, fetch  
close)

DBMS\_OUTPUT.PUT\_LINE('DEPTNO:' ||

C1\_rec.DEPTNO ||

' , DNAME:' || C1\_rec.DNAME  
|| ' , LOC : ' || C1\_rec.LOC);

END LOOP;

END;

↓  
(실행 결과)

/

동일

\* 커서 각 행을 C1\_rec 주프 인덱스에 저장하므로  
결과 행을 저장하는 변수 선언도 필요하지 않다.

(35) (18/123)

(커서에 파라미터 사용하기)

\* 사용법

CURSOR 커서이름(파라미터 이름처럼, ...) IS  
SELECT ...

DECLARE

V-DEPT-ROW DEPT%ROWTYPE;

CURSOR C1 (P-deptno DEPT. DEPTNO%TYPE) IS  
SELECT DEPTNO, DNAME, LOC  
FROM DEPT  
WHERE DEPTNO = P-deptno;

BEETIN

OPEN C1 (10);

LOOP

FETCH C1 INTO V-DEPT-ROW;  
EXIT WHEN C1%NOTFOUND;  
DBMS\_OUTPUT.PUT\_LINE(  
'10번 부서 - DEPTNO : ' || V-DEPT-ROW.DEPTNO  
' , DNAME : ' || V-DEPT-ROW.DNAME  
' , LOC : ' || V-DEPT-ROW.LOC));

END LOOP;

CLOSE C1;

OPEN C1 (20);

LOOP

FETCH C1 INTO V-DEPT-ROW;  
EXIT WHEN C1%NOTFOUND;  
DBMS\_OUTPUT.PUT\_LINE(  
'20번 부서 - DEPTNO : ' || V-DEPT-ROW.DEPTNO  
' , DNAME : ' || V-DEPT-ROW.DNAME  
' , LOC : ' || V-DEPT-ROW.LOC));

END LOOP;

CLOSE C1;

END;

↓ (결과)

10번부서 - DEPTNO:10, DNAME: ~, LOC: ~  
20번부서 - 11!20, 11 ~ ~, ~

(파라미터를 사용자에게 직접 입력받고 싶다면)

DECLARE

V-deptno DEPT. DEPTNO%TYPE;

CURSOR C1 (P-deptno DEPT. DEPTNO%TYPE) IS  
SELECT DEPTNO, DNAME, LOC  
FROM DEPT  
WHERE DEPTNO = P-deptno;

BEETIN

V-deptno := &INPUT\_DEPTNO;

(&INPUT\_DEPTNO에 브레이브 입력 받고 V-deptno에 대입)

FOR C1-rec IN C1 (V-deptno) LOOP  
DBMS\_OUTPUT.PUT\_LINE(  
'DEPTNO : ' || C1-rec.DEPNTO  
' , DNAME : ' || C1-rec.DNAME  
' , LOC : ' || C1-rec.LOC);

END LOOP;

END;

↓ (결과하면)

Input\_deptno의 값을 입력해십시오 : 10

구 : V-deptno := &INPUT\_DEPTNO;

신 : V-deptno := 10;

DEPTNO:10, DNAME: ACCOUNTING, LOC: ~

그 이외에도

✓ FOR UPDATE 절

→ 커서 안의 행이 다른 세션에 의해 변경되지 않도록  
잠금(CLOCK) 기능을 제공

✓ WHERE CURRENT OF 절

→ 커서를 통해 추출한 행에 DML 명령어를  
사용하는

(36) (18/123)

• 묵시적 (implicit) 커서

: 별다른 선언 없이 SQL문을 사용했을 때  
오라클에서 자동으로 선언되는 커서  
따라서 사용자가 OPEN, FETCH, CLOSE  
를 거치지 않습니다.

卷之三

- ✓ SQL%NOTFOUND : 추출한 행이 있으면 false  
없으면 true  
DML 명령어로 영향을 받는  
행이 없는 경우 true
  - ✓ SQL%FOUND : 추출한 행이 있으면 true  
없으면 false  
DML 명령어로 영향받는  
행이 있다면 true
  - ✓ SQL%ROWCOUNT : 추출한 행 수 또는  
DML 명령어로 영향받는  
행 수를 반환
  - ✓ SQL%ISOPEN : 자동으로 SQL문을 실행한 후  
CLOSE 되었을 때 이 속성은  
항상 false를 반환

(목시점 거시 속성 사용하기)

```
BEGIN
 UPDATE DEPT SET DNAME = 'DATABASE'
 WHERE DEPTNO = 50;
```

DBMS\_OUTPUT.PUT\_LINE('개신된 행의 수:  
|| SQL%ROWCOUNT);

```
IF (SQL%FOUND) THEN
 DBMS_OUTPUT.PUT_LINE('갱신 대상 행 존재여부: true');
ELSE
 DBMS_OUTPUT.PUT_LINE ('갱신 대상 행 존재여부: false');
END IF;
```

```
IF (SQL%ISOPEN) THEN
 DBMS_OUTPUT.PUT_LINE(
 '커서의 OPEN 여부 : true');
ELSE
 DBMS_OUTPUT.PUT_LINE(
 '커서의 OPEN 여부 : false');
END IF;
```

END ;

10

## IV(실행결과)

，정성의 ~~기록~~.

→ 생성된 행의 수 : 0  
→ 정신 대상 행 존재 여부 : false  
→ 커서의 OPEN 여부 : false

부모님께서는  
자주 open, close  
되므로 따로 키워야

< 오류가 발생해도 프로그램이 비정상 종료되지 않도록 하는 예외 처리 >

오류

```

graph LR
 A[오류] --> B[컴파일 오류
: 문법이 잘못되었거나 오타로 인한 오류]
 A --> C[런타임 오류
: 명령문의 실행 중 발생한 오류]

```

즉, 런타임 오류, 실행오류를 '예외(exception)'이라고 한다.

(예외 발생하는 PL/SQL)

**DECLARE**

V. wrong NUMBER!

BEETIN

~~SELECT DNAME INTO V\_WRONG  
FROM DEPT~~

WHERE DEPTNO = 10);

END;

③ (18/11/28)

(예외처리 추가)

DECLARE

v\_wrong NUMBER;

BEGIN

SELECT DNAME INTO v\_wrong  
FROM DEPT  
WHERE DEPTNO = 10;

EXCEPTION

WHEN VALUE\_ERROR THEN  
DBMS\_OUTPUT.PUT\_LINE

(예외처리 : 수치 또는 값 오류 발생);

END;

/      ↓

예외처리 : 수치 또는 값 오류 발생

\* 예외가 발생되면 해당 내용만 출력되고  
해당되지 않은 예외종류와 오류작성파일은  
출력되지 않는다.

- 예외 종류
    - (473~49) 내부 예외 < 이름 있는 (internal exceptions)
    - 사용자 정의 예외 (user-defined exceptions)
- ↓ (테이블)
- 이름 있는
- ✓ VALUE\_ERROR : 수치 아 값 오류
  - ✓ TOO\_MANY\_ROWS : 높고보다 많은 행 추출 오류
  - ⋮

## ○ 예외 처리부 작성

\* 방법

EXCEPTION

WHEN 예외 이름 THEN  
예외처리에 사용할 명령어;

WHEN 예외 이름2 THEN  
예외처리에 사용할 명령어;

WHEN OTHERS THEN 예외처리에 사용됨;

(이름 없는 예외 사용)

\* 방법

DECLARE

예외 이름 EXCEPTION;

PRAGMA EXCEPTION\_INIT

(예외 이름, 예외 번호);

⋮  
⋮

EXCEPTION

WHEN 예외 이름 THEN

예외처리에 사용할 명령어

⋮  
⋮

END;

(사용자 정의 예외 사용)

\* 방법

DECLARE

사용자 예외 이름 EXCEPTION;

⋮  
⋮

BEGIN

IF 사용자 예외를 발생시킬 조건 THEN

RAISE 사용자 예외 이름

⋮  
⋮

END IF;

EXCEPTION

WHEN 사용자 예외 이름 THEN

예외 처리에 사용할 명령어;

⋮  
⋮

END;

(오류 코드와 오류 메시지 사용)

함수

SQLCODE

SQLERRM

설명

오류 번호를 반환하는 함수

오류 메시지를 반환하는 함수

예시 (477 P)

(38) (18/11/24)

## <저장 서브프로그램>

### 저장 서브프로그램이란? (Stored subprogram)

: 이렇게 여러번 사용할 목적으로 이름을 지정하여 오라클에 저장해 두는 PL/SQL 프로그램을 저장 서브프로그램 이라 한다.

\* 기존의 PL/SQL 블록 (anonymous block)은 한 번 실행한 뒤 다시 실행하려면 다시 작성.

### 저장 서브프로그램 종류

- 프로시저 stored procedure
- 함수 stored function
- 패키지 package
- 트리거 trigger

### 프로시저, stored procedure

: 특정 처리 작업을 처리하는데

SQL문에서 사용할 수 있고

파라미터를 사용할 수 있고 사용하지 않을 수도 있다.

(파라미터를 사용하지 않는 프로시저)

\* 사용법.

```
CREATE [OR REPLACE] PROCEDURE 프로시저 이름
IS or AS ↳ 똑같은 이름 프로시저 존재해
 있어 쓴다는 의미로 사용.
BEGIN
EXCEPTION
END [프로시저 이름];
```

### 1) 프로시저 생성하기

```
CREATE OR REPLACE PROCEDURE pro_no_param
IS
 v_EMPNO NUMBER(4) := 7788;
 vENAME VARCHAR2(10);
BEGIN
 vENAME := 'SCOTT';
 DBMS_OUTPUT.PUT_LINE('v_EMPNO : ' || vEMPNO);
 DBMS_OUTPUT.PUT_LINE('vENAME : ' || vENAME);
END;
```

↓

프로시저가 생성되었습니다

### 2) 생성한 프로시저 실행하기

SET SERVEROUTPUT ON;

EXECUTE pro\_no\_param;

(결과하면)

v-EMPNO : 7788

v-ENAME : SCOTT

### 3) 익명 블록에서 프로시저 실행하기 (PL/SQL)

BEGIN

pro\_no\_param;

END;

/ (결과하면) 동일

### 4) 프로시저 내용 확인하기

SELECT \*

FROM USER\_SOURCE

WHERE NAME = 'PRO\_NOPARAM';

### 5) 프로시저 삭제하기

DROP PROCEDURE PRO\_NOPARAM;

(파라미터를 사용하는 프로시저)

### 6) 파라미터를 지정할 때 사용하는 모드

1) IN : 지정하지 않으면 기본값, 프로시저 호출할 때 입력받습니다.

2) OUT : 호출할 때 값을 반환합니다.

3) IN OUT : 호출할 때 값을 입력 받은 후 실행 결과 값을 반환합니다.

### 1) IN 모드 파라미터

① 프로시저에 파라미터 지정하기

```
CREATE OR REPLACE PROCEDURE pro_param_in
(
 param1 IN NUMBER, ↳ 직접 입력받는 파라미터 지정
 param2 NUMBER, (기본값이라 생각 가능)
 param3 NUMBER := 3,
 param4 NUMBER DEFAULT 4
)
```

IS

↓ 계속

③ (18/11/24)

↓ 제작

```
BEGIN
 DBMS_OUTPUT.PUT_LINE('param1 : ' || param1);
 DBMS_OUTPUT.PUT_LINE('param2 : ' || param2);
 DBMS_OUTPUT.PUT_LINE('param3 : ' || param3);
 DBMS_OUTPUT.PUT_LINE('param4 : ' || param4);
END;
```

↓  
프로시저 생성 완료

② 파라미터 입력하여 프로시저 사용

```
EXECUTE pro-param-in(1, 2, 9, 8);
 ^
 2
 9
 8
```

③ 기본값이 지정된 파라미터 입력을 제외하고 프로시저 사용

```
EXECUTE pro-param-in(1, 2);
 ^
 2
 3
 4
```

④ 실행에 필요한 개수보다 적은 파라미터를  
입력하여 프로시저 실행하기

```
EXECUTE pro-param-in(1);
 ^
 오류발생
```

⑤ 파라미터 이름을 활용하여 프로시저에 값 입력

```
EXECUTE pro-param-in(
 param1 => 10, param2 => 20);
 ^ 10 20 ↗ 이름지정시 필요한 연산자.
 | 3 4
```

## 2) OUT 보드 파라미터

① OUT 보드 파라미터 정의하기

```
CREATE OR REPLACE PROCEDURE pro-param-out
(
 in_empno IN EMP.EMPNO%TYPE,
 out_ename OUT EMP.ENAME%TYPE,
 out_sal OUT EMP.SAL%TYPE
)
IS
```

BEGIN

```
 SELECT ENAME, SAL INTO out_ename,
 out_sal
 FROM EMP
 WHERE EMPNO = in_empno;
END pro-param-out;
```

② OUT 보드 파라미터 사용하기

DECLARE

```
 v_ename EMP.ENAME%TYPE;
 v_sal EMP.SAL%TYPE;
```

BEGIN

```
 pro-param-out(1188, v_ename, v_sal);
 DBMS_OUTPUT.PUT_LINE('ENAME : ' || v_ename);
 DBMS_OUTPUT.PUT_LINE('SAL : ' || v_sal);
```

END;

↓  
ENAME : SCOTT  
SAL : 3000

## 3) IN OUT 보드 파라미터

① IN OUT 보드 파라미터 정의하기

```
CREATE OR REPLACE PROCEDURE pro-param-inout
(
 inout_no IN OUT NUMBER
)
IS
```

BEGIN

```
 inout_no := inout_no * 2;
```

```
END pro-param-inout;
```

↓

② IN OUT 보드 파라미터 사용하기

DECLARE

no NUMBER;

BEGIN

n := 3;

pro-param-inout(no);

DBMS\_OUTPUT.PUT\_LINE('no : ' || no);

END;

↓

no : 10

④ (18/11/24)

4) 오류 확인 방법  
(493P)

- SHOW ERRORS  
(SQL\*PLUS 미만)
- USER-ERRORS  
(토드같은 유용 프로그램에서)

## ○ 함수

1) 함수 생성하기

```
CREATE OR REPLACE FUNCTION func_aftertax(
 sal IN NUMBER
)
```

```
 RETURN NUMBER; → 함수(실행후 반환하는
 자료형)
 IS
 tax NUMBER := 0.05;
```

```
BEGIN
 RETURN (ROUND (sal - (sal * tax)));
END func_aftertax;
```

2) PL/SQL에서 함수 실행

```
DECLARE
 aftertax NUMBER;
BEGIN
 aftertax := func_aftertax(3000);
 DBMS_OUTPUT.PUT_LINE('after-tax income: ' ||
 aftertax);
END;
```

↓  
after-tax income: 2850

3) SQL에서 함수 실행

```
SELECT func_aftertax(3000)
FROM DUAL;
```

↓  
FUNC-AFTERTAX(3000)

4) 함수에 테이블 데이터 사용하기

```
SELECT EMPNO, ENAME, SAL,
 func_aftertax(SAL) AS AFTERTAX
 FROM EMP;
```

5) 함수 삭제하기

```
DROP FUNCTION func_aftertax;
```

○ 패키지 (package)

• 여러개의 PL/SQL 서비스 프로그램을 하나의  
 논리 그룹으로 묶어 통합 관리하는데 사용하는  
 객체를 뜻합니다.

1) 패키지 → 조각 생성

↳ 두 부분으로 나누어 제작

명세  
(specification)

본문  
(body)

- 패키지 명세 생성 (501P)

```
CREATE [OR REPLACE] PACKAGE 패키지 이름
IS [or AS]
 서비스 프로그램을 포함한 다양한 객체 선언
END [패키지 이름];
```

- 패키지 본문 생성 (502P)

```
CREATE [OR REPLACE] PACKAGE 패키지 이름
IS : AS
 패키지 명세에서 선언한 서비스 프로그램을
 포함한 여러 객체를 정의
 경우에 따라 패키지 명세에 존재하지 않는
 객체 및 서비스 프로그램도 정의 가능.
END [패키지 이름];
```

④ (18/11/24)

## 2) 서브프로그램 오버로드 (504P)

: 기본적으로 서브프로그램 이름은 충복될수 있음  
하지만 같은 패키지에서 사용하는  
파라미터의 개수, 자료형, 순서가 다를 경우에  
한해서만 이름이 같은 서브프로그램을  
정의할 수 있습니다. 이를 서브프로그램  
오버로드라고 합니다.

CREATE [OR REPLACE] PACKAGE 패키지 이름  
IS or AS

서브프로그램 종류 서브프로그램 이름(파라미터 정의);  
서브프로그램 종류 서브프로그램 이름 (  
개수나 자료형, 순서가 다른 파라미터 정의);  
END [패키지 이름];

## 3) 패키지 사용하기 (506P)

### 4) 패키지 삭제하기 (507P)

(명세 & 본문 한 번에 삭제)

DROP PACKAGE 패키지 이름;

(패키지의 본문만을 삭제)

DROP PACKAGE BODY 패키지 이름;

## ◦ 트리거 (trigger)

: 데이터베이스 안의 특정 상황이나 동작,  
즉 이벤트가 발생할 경우에 자동으로  
실행되는 기능을 정의하는 PL/SQL 서브프로그램.

◦ DML트리거 : INSERT, UPDATE, DELETE

◦ DPL트리거 : CREATE, ALTER, DROP

◦ INSTEAD OF 트리거 : view

◦ 시그닝 트리거 : 데이터베이스나 소프트웨어로 동작

◦ 단순트리거 :

◦ 복합트리거 :

## 1) DML 트리거

CREATE [OR REPLACE] TRIGGER 트리거 이름  
BEFORE or AFTER

INSERT or UPDATE or DELETE ON 테이블 이름  
REFERENCING OLD AS old 와

    VIEW AS new  
FOR EACH ROW WHEN 조건식

FOLLOWS 트리거 이름2, 트리거 이름3 ...

ENABLE or DISABLE

DECLARE

시작부

BEGIN

실행부

EXCEPTION

예외 처리부

END;

(EMP-TRGT 테이블 생성하기)

CREATE TABLE EMP-TRGT

AS SELECT \* FROM EMP;

(DML - 실행 전에 수행할 트리거 생성하기)

CREATE OR REPLACE TRIGGER

trg\_empnodml\_weekend

BEFORE

INSERT OR UPDATE OR DELETE ON EMP-TRGT

BEGIN

IF TO\_CHAR(SYSDATE, 'DD') IN ('토', '일') THEN

IF INSERTING THEN

raise\_application\_error (-20000,

'주말 사원정보 추가 불가');

ELSIF UPDATING THEN

raise\_application\_error (-20001,

'주말 사원정보 수정 불가');

ELSIF DELETING THEN

raise\_application\_error (-20002,

'주말 사원정보 삭제 불가');

ELSE

raise\_application\_error (-20003,

'주말 사원정보 변경 불가');

END IF;

END IF;

END;

④ (18/124)

(평일 날짜로 EMP\_TRG 테이블 UPDATE하기)

```
UPDATE emp_trg SET sal =
 3500 WHERE empno = 7788;
```

(주말 날짜로 EMP\_TRG 테이블 UPDATE하기)

```
UPDATE emp_trg SET sal = 3500
WHERE empno = 7788;
```

## 2) DML 트리거의 제작 및 사용 (AFTER)

(EMP\_TRG\_LOG 테이블 생성하기)

```
CREATE TABLE EMP_TRG_LOG(
 TABLENAME VARCHAR2(10),
 DML_TYPE VARCHAR2(10),
 EMPNO NUMBER(4),
 USER_NAME VARCHAR2(30),
 CHANGE_DATE DATE
)
```

(DML 실행 후 수행할 트리거 생성하기) (5/3P)

```
CREATE OR REPLACE TRIGGER trg_emp_log
AFTER
INSERT OR UPDATE OR DELETE ON EMP_TRG
FOR EACH ROW
BEGIN
```

```
IF INSERTING THEN
 INSERT INTO emp_trg_log
 VALUES ('EMP_TRG', 'INSERT', :new.empno,
 SYS_CONTEXT('USERENV', 'SESSION_USER'),
 SYSDATE);
```

```
ELSIF UPDATING THEN
 INSERT INTO emp_trg_log
 VALUES ('EMP_TRG', 'UPDATE', :old.empno,
 SYS_CONTEXT('USERENV', 'SESSION_USER'),
 SYSDATE);
```

```
ELSIF DELETING THEN
 INSERT INTO emp_trg_log
 VALUES ('EMP_TRG', 'DELETE', :old.empno,
 SYS_CONTEXT('USERENV', 'SESSION_USER'),
 SYSDATE);
```

END IF;

END;

(EMP\_TRG 테이블에 INSERT 실행하기)

```
INSERT INTO EMP_TRG
VALUES (9999, 'testEmp', 'CLERK', 7788,
 TO_DATE('2018-03-03', 'YYYY-MM-DD'),
 1200, null, 20);
```

(EMP\_TRG 테이블에 INSERT 실행해 - COMMIT하기)

COMMIT;

(EMP\_TRG 테이블의 INSERT 확인하기)

```
SELECT *
FROM EMP_TRG;
```

(EMP\_TRG\_LOG 테이블의 INSERT 기록 확인하기)

```
SELECT *
FROM EMP_TRG_LOG;
```

(EMP\_TRG 테이블에 UPDATE 실행하기)

```
UPDATE EMP_TRG
SET SAL = 1300
WHERE MGR = 7788;
```

(EMP\_TRG 테이블에 UPDATE 실행하기 - COMMIT하기)

COMMIT;

## 3) 트리거 관리

(트리거 정보 조회)

```
SELECT TRIGGER_NAME, TRIGGER_TYPE,
 TRIGGERING_EVENT, TABLE_NAME,
 STATUS
 FROM USER_TRIGGERS;
```

## 4) 트리거 변경

```
ALTER TRIGGER 트리거 이름 ENABLE
 or DISABLE;
```

## 5) 트리거 삭제

```
DROP TRIGGER 트리거 이름;
```