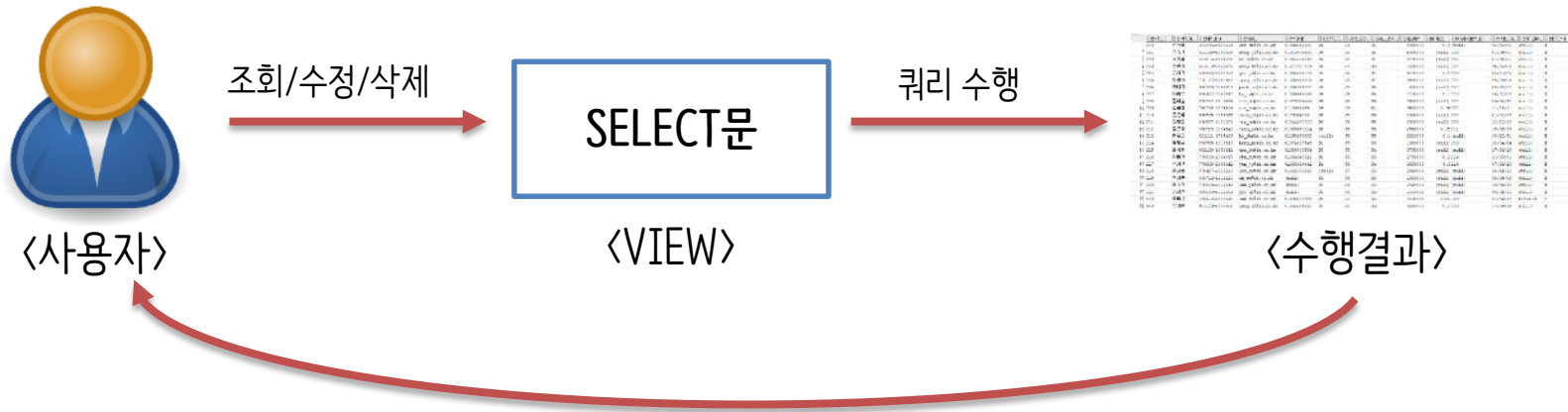


ORACLE OBJECT(VIEW)

KH KH정보교육원

VIEW란?

SELECT 쿼리의 실행 결과를 화면에 저장한 논리적인 가상 테이블이다. 테이블과는 다르게 실질적으로 데이터를 저장하고 있지 않지만, 사용자는 테이블을 사용하는 것과 동일하게 사용할 수 있다.



Oracle Object - VIEW

VIEW란?

```
CREATE OR REPLACE VIEW V_EMPLOYEE(사번, 이름, 부서, 지역)
AS SELECT EMP_ID, EMP_NAME, DEPT_TITLE,
NATIONAL_NAME
FROM EMPLOYEE
LEFT JOIN DEPARTMENT ON (DEPT_ID = DEPT_CODE)
LEFT JOIN LOCATION ON (LOCATION_ID = LOCAL_CODE)
LEFT JOIN NATIONAL USING (NATIONAL_CODE);
```

```
SELECT * FROM V_EMPLOYEE;
```

SQL | 인출된 모든 행: 24(0초)

	사번	이름	부서	지역
1	900	장채현	인사관리부	한국
2	217	전지연	인사관리부	한국
3	216	차태연	인사관리부	한국
4	214	방명수	인사관리부	한국
5	221	유하진	회계관리부	한국
6	220	이중석	회계관리부	한국
7	219	임시환	회계관리부	한국
8	202	노용철	총무부	한국
9	201	송종기	총무부	한국
10	200	선동일	총무부	한국
11	215	대복훈	해외영업1부	일본
12	210	윤은해	해외영업1부	일본
13	209	심봉선	해외영업1부	일본
14	208	김해솔	해외영업1부	일본
15	207	하이유	해외영업1부	일본
16	206	박나라	해외영업1부	일본
17	205	정중하	해외영업2부	중국
18	204	유재식	해외영업2부	중국
19	203	송은희	해외영업2부	중국
20	222	이태림	기술지원부	러시아
21	212	장프위	기술지원부	러시아
22	211	전형돈	기술지원부	러시아
23	218	미오리	(null)	(null)
24	213	하동운	(null)	(null)

Oracle Object - VIEW

VIEW란?

```
CREATE OR REPLACE VIEW V_EMP_JOB(사번, 이름, 직급, 성별, 근무년수)
AS SELECT EMP_ID,
        EMP_NAME,
        JOB_NAME,
        DECODE(SUBSTR(EMP_NO, 8, 1), 1, '남', 2, '여'),
        EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM HIRE_DATE)
FROM EMPLOYEE
JOIN JOB USING(JOB_CODE);
```

** 서브쿼리의 SELECT절에 함수가 사용된 경우 반드시 별칭을 지정해 주어야 한다.

SQL | 인출된 모든 행: 24(0초)

	사번	이름	직급	성별	근무년수
1	200	선동일	대표	남	27
2	202	노웅철	부사장	남	16
3	201	송종기	부사장	남	16
4	209	심봉선	부장	남	6
5	205	정중하	부장	남	18
6	204	유재식	부장	남	17
7	221	유하진	차장	남	23
22	210	윤은혜	사원	여	16
23	206	박나라	사원	여	9
24	900	장채현	(null)	남	0

Oracle Object - VIEW

VIEW란?

```
CREATE OR REPLACE VIEW V_JOB  
AS SELECT JOB_CODE, JOB_NAME  
FROM JOB;
```

	JOB_CODE	JOB_NAME
1	J1	대표
2	J2	부사장
3	J3	부장
4	J4	차장
5	J5	과장
6	J6	대리
7	J7	사원

```
INSERT INTO V_JOB VALUES('J8', '인턴');
```

** 생성된 뷰를 가지고 DML구문(INSERT, UPDATE, DELETE) 사용 가능하다.

** 생성된 뷰에 요청한 DML구문이 베이스 테이블도 변경한다.

```
SELECT * FROM V_JOB;
```

	JOB_CODE	JOB_NAME
1	J1	대표
2	J2	부사장
3	J3	부장
4	J4	차장
5	J5	과장
6	J6	대리
7	J7	사원
8	J8	인턴

```
SELECT * FROM JOB;
```

	JOB_CODE	JOB_NAME
1	J1	대표
2	J2	부사장
3	J3	부장
4	J4	차장
5	J5	과장
6	J6	대리
7	J7	사원
8	J8	인턴

Oracle Object - VIEW

DML 명령어로 조작이 불가능한 경우

가능

~~insert update delete~~

1. 뷰 정의에 포함되지 않은 컬럼을 조작하는 경우
2. 뷰에 포함되지 않은 컬럼 중에 베이스가 되는 테이블 컬럼이 NOT NULL 제약조건이 지정된 경우
3. 산술 표현식으로 정의된 경우
4. JOIN을 이용해 여러 테이블을 연결한 경우
5. DISTINCT를 포함한 경우
6. 그룹함수나 GROUP BY 절을 포함한 경우

관련 데이터

불가능

Oracle Object - VIEW

1. 뷰 정의에 포함되지 않은 컬럼을 조작하는 경우

```
CREATE OR REPLACE VIEW V_JOB2  
AS SELECT JOB_CODE  
FROM JOB;
```

```
INSERT INTO V_JOB2 VALUES('J8', '인턴');
```

- ** 뷰 정의에 포함되지 않은 컬럼을 INSERT/UPDATE하는 경우 에러 발생
- ** 단, DELETE는 가능하다.

	JOB_CODE
1	J1
2	J2
3	J3
4	J4
5	J5
6	J6
7	J7

오류 보고 -

SQL 오류: ORA-00913: too many values

00913. 00000 - "too many values"

*Cause:

*Action:

2. 뷰에 포함되지 않은 컬럼 중에 베이스가 되는 테이블 컬럼이 NOT NULL 제약조건이 지정된 경우

```
CREATE OR REPLACE VIEW V_JOB3  
AS SELECT JOB_NAME  
FROM JOB;
```

```
INSERT INTO V_JOB3 VALUES('인턴');
```

- ** 뷰에 포함되지 않은 NOT NULL 제약조건이 있는 컬럼이 존재하면 INSERT/UPDATE시 에러 발생
- ** 단, DELETE는 가능하다.

	JOB_NAME
1	대표
2	부사장
3	부장
4	차장
5	과장
6	대리
7	사원

오류 보고 -

SQL 오류: ORA-01400: cannot insert NULL into ("EMPLOYEE"."JOB"."JOB_CODE")

01400. 00000 - "cannot insert NULL into (%s)"

*Cause: An attempt was made to insert NULL into previously listed objects.

*Action: These objects cannot accept NULL values.

Oracle Object - VIEW

3. 산술 표현식으로 정의된 경우

```
CREATE OR REPLACE VIEW EMP_SAL
AS SELECT EMP_ID,
        EMP_NAME,
        SALARY,
        (SALARY + (SALARY * NVL(BONUS, 0))) * 12 연봉
FROM EMPLOYEE;
```

```
INSERT INTO EMP_SAL
VALUES(800, '정진훈', 3000000, 4000000);
```

- ** 뷰에 산술 계산식이 포함된 경우 INSERT/UPDATE시 에러 발생
- ** 단, DELETE는 가능하다.

SQL | 인출된 모든 행: 24(0.016초)

EMP_ID	EMP_NAME	SALARY	연봉
1 200	선동일	8000000	124800000
2 201	송중기	6000000	72000000
3 202	노웅철	3700000	44400000
4 203	송은희	2800000	33600000
5 204	유재석	3400000	40800000
6 205	정중하	3900000	46800000
7 206	박나라	1800000	21600000
8 207	하미유	2200000	26400000
9 208	김해솔	2500000	30000000
10 209	심봉선	3500000	42000000
11 210	윤은혜	2000000	24000000
12 211	전철도	2000000	24000000
13 212	장프위	2550000	30600000
14 213	하동운	2320000	27840000
15 214	방명수	1380000	16560000
16 215	대복훈	3760000	45120000
17 216	차태연	2780000	33360000
18 217	전지연	3660000	43920000
19 218	미오리	2890000	34680000
20 219	임시환	1550000	18600000
21 220	이중석	2490000	29880000
22 221	유하진	2480000	29760000
23 222	미태림	2436240	29234880
24 900	장채현	4300000	51600000

오류 보고 -

SQL 오류: ORA-01733: virtual column not allowed here
01733. 00000 - "virtual column not allowed here"

*Cause:

*Action:

Oracle Object - VIEW

4. JOIN을 이용해 여러 테이블을 연결한 경우

```
CREATE OR REPLACE VIEW V_JOINEMP
AS SELECT EMP_ID,
           EMP_NAME,
           DEPT_TITLE
FROM EMPLOYEE
JOIN DEPARTMENT ON (DEPT_CODE = DEPT_ID);
```

EMP_ID	EMP_NAME	DEPT_TITLE
1 200	선동일	총무부
2 201	송중기	총무부
3 202	노웅철	총무부
4 203	송은희	해외영업2부
5 204	유재식	해외영업2부
6 205	정중하	해외영업2부
7 206	박나라	해외영업1부
8 207	하미유	해외영업1부
9 208	김해솔	해외영업1부
10 209	심봉선	해외영업1부
11 210	윤은혜	해외영업1부
12 211	전형도	기술지원부
13 212	장프위	기술지원부
14 214	방명수	인사관리부
15 215	대복훈	해외영업1부
16 216	차태연	인사관리부
17 217	전지연	인사관리부
18 219	임시환	회계관리부
19 220	이종석	회계관리부
20 221	유하진	회계관리부
21 222	이태림	기술지원부
22 900	장채현	인사관리부

```
INSERT INTO V_JOINEMP
VALUES (888, '조세오', '인사관리부');
```

- ** 뷰 정의시 JOIN을 사용한 경우 INSERT/UPDATE시 에러 발생
- ** 단, DELETE는 가능하다.

오류 보고 -

SQL 오류: ORA-01776: cannot modify more than one base table through a join view
01776. 00000 - "cannot modify more than one base table through a join view"

*Cause: Columns belonging to more than one underlying table were either inserted into or updated.

*Action: Phrase the statement as two or more separate statements.

5. DISTINCT를 포함한 경우

```
CREATE OR REPLACE VIEW V_DT_EMP  
AS SELECT DISTINCT JOB_CODE  
FROM EMPLOYEE;
```

```
INSERT INTO V_DT_EMP VALUES('J9');
```

** DISTINCT를 사용한 경우 DML 사용 불가

```
DELETE FROM V_DT_EMP  
WHERE JOB_CODE = 'J1';
```

** DELETE시에도 에러 발생

JOB_CODE
1 J2
2 J7
3 J3
4 J6
5 J5
6 J8
7 J1
8 J4

오류 보고 -

SQL 오류: ORA-01732: data manipulation operation not legal on this view
01732. 00000 - "data manipulation operation not legal on this view"
*Cause:
*Action:

오류 보고 -

SQL 오류: ORA-01732: data manipulation operation not legal on this view
01732. 00000 - "data manipulation operation not legal on this view"
*Cause:
*Action:

Oracle Object - VIEW

6. 그룹함수나 GROUP BY 절을 포함한 경우

```
CREATE OR REPLACE VIEW V_GROUPDEPT
AS SELECT DEPT_CODE,
          SUM(SALARY) 합계,
          AVG(SALARY) 평균
FROM EMPLOYEE
GROUP BY DEPT_CODE;
```

```
INSERT INTO V_GROUPDEPT
VALUES('D10', 6000000, 4000000);
```

** 그룹함수 혹은 GROUP BY를 사용한 경우
INSERT/UPDATE시 에러 발생

```
DELETE FROM V_GROUPDEPT
WHERE DEPT_CODE = 'D1';
```

** DELETE시에도 에러 발생

DEPT_CODE	합계	평균
1 (null)	5210000	2605000
2 D1	12120000	3030000
3 D9	17700000	5900000
4 D5	15760000	2626666
5 D6	10100000	3366666
6 D2	6520000	2173333
7 D8	6986240	2328746

오류 보고 -

SQL 오류: ORA-01733: virtual column not allowed here

01733. 00000 - "virtual column not allowed here"

*Cause:

*Action:

오류 보고 -

SQL 오류: ORA-01732: data manipulation operation not legal on this view

01732. 00000 - "data manipulation operation not legal on this view"

*Cause:

*Action:

Oracle Object - VIEW

```
SELECT * FROM USER_VIEWS;
```

VIEW_NAME	TEXT_LENGTH	TEXT	TYPE_TEXT_LENGTH	TYPE_TEXT	OID_TEXT_LENGTH	OID_TEXT	VIEW_TYPE_OWNER	VIEW_TYPE	SUPERVIE
1 V_EMPLOYEE	189	SELECT EMP_ID, EMP_NAME, DEPT_TITLE, NATIONAL_NA...	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
2 V_EMP_JOB	226	SELECT EMP_ID, EMP_NAME, JOB_N...	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
3 V_EMP	162	SELECT "EMP_ID", "EMP_NAME", "EMP_NO", "EMAIL", "PHO...	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
4 V_JOB	34	SELECT JOB_CODE, JOB_NAMEFROM JOB	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
5 V_JOB2	24	SELECT JOB_CODEFROM JOB	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
6 V_JOB3	24	SELECT JOB_NAMEFROM JOB	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
7 V_DT_EMP	38	SELECT DISTINCT JOB_CODE FROM EMPLOYEE	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
8 V_GROUPDEPT	101	SELECT DEPT_CODE, SUM(SALARY) 합계, FLOOR(AVG(SAL...	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
9 EMP_SAL	93	SELECT EMP_ID, EMP_NAME, SALARY, (SALARY + (SALA...	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)
10 V_JOINEMP	90	SELECT EMP_ID, EMP_NAME, DEPT_TITLEFROM EMPLOYEE...	(null) (null)	(null) (null)	(null) (null)	(null) (null)	(null)	(null)	(null)

** 뷰 정의시 사용한 쿼리 문장이 TEXT 컬럼에 저장되어 있음.

** 뷰가 재생 될 때는 TEXT에 기록된 SELECT에 문장이 다시 실행되면서 결과를 보여주는 구조이다.

VIEW 옵션

1. OR REPLACE 옵션

- 생성한 뷰가 존재하면, 뷰를 갱신한다.

2. FORCE/NOFORCE 옵션

- FORCE 옵션은 기본 테이블이 존재하지 않더라도 뷰를 생성한다.
- 기본값은 NOFORCE로 지정되어 있다.

3. WITH CHECK OPTION 옵션

- 옵션을 설정한 컬럼의 값을 수정하지 못하게 한다.

4. WITH READ ONLY 옵션

- 뷰에 대해 조회만 가능하고, 삽입, 수정, 삭제 등을 하지 못하게 한다.

Oracle Object - VIEW

INLINE VIEW

일반적으로 FROM 절에 사용된 서브쿼리의 결과 화면에 별칭을 붙인 것을 말한다.
FROM절에 서브쿼리를 직접 사용해도 되고, 따로 뷰를 생성 후 FROM절에서 생성한 뷰를 사용해도 된다.

ORACLE OBJECT(SEQUENCE)

SEQUENCE란?

순차적으로 정수 값을 자동으로 생성하는 객체로, 자동 번호 발생기의 역할을 한다.

표현식

CREATE SEQUENCE 시퀀스명

- ① [START WITH 숫자]
- ② [INCREMENT BY 숫자]
- ③ [MAXVALUE 숫자 | NOMAXVALUE]
- ④ [MINVALUE 숫자 | NOMINVALUE]
- ⑤ [CYCLE | NOCYCLE]
- ⑥ [CACHE | NOCACHE]

- ① 처음 발생시킬 시작 값 지정, 기본값 1
- ② 다음 값에 대한 증가치, 기본값 1
- ③ 발생시킬 최대값 지정, 10의 27승-1까지 가능
- ④ 발생시킬 최소값 지정, -10의 26승
- ⑤ 시퀀스가 최대값까지 증가를 완료하면 CYCLE은 START WITH 설정값으로 돌아가고, NOCYCLE은 에러 발생
- ⑥ CACHE는 메모리상에서 시퀀스값을 관리, 기본값 20

Oracle Object - SEQUENCE

SEQUENCE 객체 생성

```
CREATE SEQUENCE SEQ_EMPID  
START WITH 300  
INCREMENT BY 5  
MAXVALUE 310  
NOCYCLE  
NOCACHE;
```

```
SELECT SEQ_EMPID.CURRVAL  
FROM DUAL;
```

ORA-08002: sequence SEQ_EMPID.CURRVAL is not yet defined in this session
08002, 00000 - "sequence %s.CURRVAL is not yet defined in this session"
*Cause: sequence CURRVAL has been selected before sequence NEXTVAL
*Action: select NEXTVAL from the sequence before selecting CURRVAL

```
SELECT SEQ_EMPID.NEXTVAL FROM DUAL;
```

NEXTVAL	
	300

```
SELECT SEQ_EMPID.CURRVAL FROM DUAL;
```

CURRVAL	
	300

```
SELECT SEQ_EMPID.NEXTVAL FROM DUAL; X 2
```

NEXTVAL	
1	305



NEXTVAL	
1	310

```
SELECT SEQ_EMPID.NEXTVAL FROM DUAL;
```

ORA-08004: sequence SEQ_EMPID.NEXTVAL exceeds MAXVALUE and cannot be instantiated
08004, 00000 - "sequence %s.NEXTVAL %s %sVALUE and cannot be instantiated"
*Cause: instantiating NEXTVAL would violate one of MAX/MINVALUE
*Action: alter the sequence so that a new value can be requested

Oracle Object - SEQUENCE

생성된 시퀀스 확인

```
SELECT * FROM USER_SEQUENCES;
```

	SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	CYCLE_FLAG	ORDER_FLAG	CACHE_SIZE	LAST_NUMBER
1	SEQ_EMPID	1	310	5	N	N	0	315

Oracle Object - SEQUENCE

NEXTVAL / CURRVAR 사용 가능 여부

사용가능	사용불가
서브쿼리가 아닌 SELECT문	VIEW의 SELECT절
INSERT문의 SELECT절	DISTINCT 키워드가 있는 SELECT문
INSERT문의 VALUE절	GROUP BY, HAVING, ORDER BY 절이 있는 SELECT문
UPDATE문의 SET절	SELECT, DELETE, UPDATE의 서브쿼리
	CREATE TABLE, ALTER TABLE 명령의 DEFAULT값

CURRVAL을 사용할때 주의 할점

시퀀스의 첫 시작값은 없으므로 NEXTVAL을 무조건 1번을 실행후에 CURRVAL을 할 수 있음

Oracle Object - SEQUENCE

SEQUENCE 예제

```
CREATE TABLE SEQ_TEST -- 테스트 테이블 생성
(
  NO NUMBER PRIMARY KEY,
  NAME NVARCHAR2(20) NOT NULL,
  AGE NUMBER NOT NULL
);
```

```
SEQ_TEST_NO.NEXTVAL; -- 생성후 초기 NEXTVAL

-- SEQ_TEST_NO 라는 시퀀스 객체 생성
CREATE SEQUENCE SEQ_TEST_NO
START WITH 1          -- 시작 번호는 1부터
INCREMENT BY 1       -- 1씩 증가
MAXVALUE 100         -- 최대 100까지
NOCYCLE              -- 100 이후에는 에러 발생
NOCACHE;             -- 캐쉬 사용 안함
```

```
INSERT INTO SEQ_TEST VALUES(SEQ_TEST_NO.NEXTVAL,'홍길동',20);
INSERT INTO SEQ_TEST VALUES(SEQ_TEST_NO.NEXTVAL,'김말똥',21);
```

Oracle Object - SEQUENCE

SEQUENCE 변경

시퀀스 수정 시 CREATE에 사용한 옵션을 변경할 수 있다.

단 START WITH값 변경은 불가능하기 때문에 변경하려면 삭제 후 다시 생성해야 한다.

```
ALTER SEQUENCE SEQ_EMPID  
INCREMENT BY 10  
MAXVALUE 400  
NOCYCLE  
NOCACHE;
```

Sequence SEQ_EMPID이 (가) 변경되었습니다.

```
SELECT * FROM USER_SEQUENCES WHERE SEQUENCE_NAME = 'SEQ_EMPID';
```

SEQUENCE 삭제

```
DROP SEQUENCE SEQ_EMPID;
```

Oracle Object - SEQUENCE

SEQUENCE 변경

시퀀스 간단 예제

KH_MEMBER 테이블을 생성
컬럼

```
MEMBER_ID    NUMBER
MEMBER_NAME  VARCHAR2(20)
MEMBER_AGE   NUMBER
MEMBER_JOIN_COM      NUMBER
```

이때 해당 사원들의 정보를 INSERT 해야 함
ID 값과 JOIN_COM은 SEQUENCE 를 이용하여 정보를 넣고자 함

ID값은 500 번 부터 시작하여 10씩 증가하여 저장 하고자 함
JOIN_COM 값은 1번부터 시작하여 1씩 증가하여 저장 해야 함
(ID 값과 JOIN_COM 값의 MAX는 10000으로 설정)

MEMBER_ID	MEMBER_NAME	MEMBER_AGE	MEMBER_JOIN_COM	
500	홍길동	20		1
510	김말똥	30		2
520	삼식이	40		3
530	고길동	24		4

ORACLE OBJECT(INDEX)

Oracle Object - INDEX *Root*

INDEX

정의

SQL 명령문의 처리 속도를 향상시키기 위해서 컬럼에 대해서 생성하는 오라클 객체이다. 내부구조는 B*트리 형식으로 구성되어 있다.

장점

검색속도가 빨라지고 시스템에 걸리는 부하를 줄여서 시스템 전체 성능을 향상시킬 수 있다.

단점




인덱스를 위한 추가 저장 공간이 필요하고, 인덱스를 생성하는데 시간이 걸린다. 따라서 데이터의 변경 작업(INSERT / UPDATE / DELETE)이 자주 일어날 경우에는 오히려 성능이 저하된다.

Oracle Object - INDEX

INDEX 표현식

CREATE [UNIQUE] INDEX 인덱스명
ON 테이블명 (컬럼명, 컬럼명 | 함수명, 함수계산식);

```
SELECT * FROM USER_IND_COLUMNS;
```

   SQL | 인출된 모든 행: 19(0.015초)

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
1 SYS_C007182	USER_UNIQUE	USER_ID	1	20	20	ASC
2 SYS_C007184	USER_UNIQUE2	USER_ID	1	20	20	ASC
3 SYS_C007186	USER_UNIQUE3	USER_ID	2	20	20	ASC
4 SYS_C007186	USER_UNIQUE3	USER_NO	1	22	0	ASC
5 SYS_C007188	USER_PRIMARYKEY	USER_NO	1	22	0	ASC
6 SYS_C007189	USER_PRIMARYKEY	USER_ID	1	20	20	ASC
7 SYS_C007196	USER_PRIMARYKEY2	USER_ID	2	20	20	ASC
8 SYS_C007196	USER_PRIMARYKEY2	USER_NO	1	22	0	ASC
9 SYS_C007210	USER_GRADE	GRADE_CODE	1	22	0	ASC
10 SYS_C007212	USER_FOREIGNKEY	USER_NO	1	22	0	ASC
11 SYS_C007213	USER_FOREIGNKEY	USER_ID	1	20	20	ASC
12 SYS_C007226	USER_CHECK	USER_NO	1	22	0	ASC
13 SYS_C007227	USER_CHECK	USER_ID	1	20	20	ASC
14 엔터티1_PK	EMPLOYEE	EMP_ID	1	3	3	ASC
15 엔터티1_PK1	JOB	JOB_CODE	1	2	2	ASC
16 엔터티1_PK2	DEPARTMENT	DEPT_ID	1	2	2	ASC
17 엔터티1_PK3	LOCATION	LOCAL_CODE	1	2	2	ASC
18 엔터티1_PK4	NATIONAL	NATIONAL_CODE	1	2	2	ASC
19 엔터티2_PK	SAL_GRADE	SAL_LEVEL	1	2	2	ASC

Oracle Object - INDEX

INDEX 구조

```
SELECT ROWID, EMP_ID, EMP_NAME  
FROM EMPLOYEE;
```

IDX_EMPID

KEY	ROWID
200	AAAE7UAABAAALC5AAA
201	AAAE7UAABAAALC5AAB
202	AAAE7UAABAAALC5AAC

AAAE7UAABAAALC5AAA

AAAE7UA

데이터
오브젝트번호

ABA

파일
번호

AALC5

BLOCK
번호

AAA

ROW
번호

ROWID	EMP_ID	EMP_NAME
1 AA AE7UAABAAALC5AAA	200	선동일
2 AA AE7UAABAAALC5AAB	201	송중기
3 AA AE7UAABAAALC5AAC	202	노용철
4 AA AE7UAABAAALC5AAD	203	송은희
5 AA AE7UAABAAALC5AAE	204	유재석
6 AA AE7UAABAAALC5AAF	205	정중하
7 AA AE7UAABAAALC5AAG	206	박나라
8 AA AE7UAABAAALC5AAH	207	하미유
9 AA AE7UAABAAALC5AAI	208	김해술
10 AA AE7UAABAAALC5AAJ	209	심봉선
11 AA AE7UAABAAALC5AAK	210	윤은해
12 AA AE7UAABAAALC5AAL	211	전형돈
13 AA AE7UAABAAALC5AAM	212	장프위
14 AA AE7UAABAAALC5AAN	213	하동운
15 AA AE7UAABAAALC5AAO	214	방명수
16 AA AE7UAABAAALC5AAP	215	대복혼
17 AA AE7UAABAAALC5AAQ	216	차태연
18 AA AE7UAABAAALC5AAR	217	전지연
19 AA AE7UAABAAALC5AAS	218	이오리
20 AA AE7UAABAAALC5AAT	219	임시환
21 AA AE7UAABAAALC5AAU	220	이종석
22 AA AE7UAABAAALC5AAV	221	유하진
23 AA AE7UAABAAALC5AAW	222	이태림
24 AA AE7UAABAAALC5AAX	900	장채현

Oracle Object - INDEX

INDEX 정보 조회

```
SELECT * FROM USER_IND_COLUMNS;
```

INDEX는 실제로 사용자가 생성하지 않게 되더라도 기본키나 유일 키 같은 제약 조건을 지정하면 자동으로 인덱스를 생성함

효율적인 사용 예

전체 데이터 중에서 10% ~ 15% 이내의 데이터를 검색하는 경우
두 개 이상의 칼럼이 WHERE절이나 조인(join) 조건으로 자주 사용되는 경우
한 번 입력된 데이터의 변경이 자주 일어나지 않는 경우
한 테이블에 저장된 데이터 용량이 상당히 클 경우

Oracle Object - INDEX

실습예제

인덱스 생성

```
CREATE INDEX EMP_IND_NND ON  
EMPLOYEE(EMP_NAME,EMP_NO,HIRE_DATE);
```

생성된 인덱스 조회

```
SELECT * FROM USER_IND_COLUMNS WHERE  
INDEX_NAME = 'EMP_IND_NND' ;
```

INDEX 생성 후 데이터 조회

```
SELECT EMP_NAME,EMP_NO,HIRE_DATE FROM EMPLOYEE;
```

인덱스 삭제

```
DROP INDEX EMP_IND_EMP_NAME ;
```

INDEX 삭제 후 데이터 조회

```
SELECT EMP_NAME,EMP_NO,HIRE_DATE FROM EMPLOYEE;
```

INDEX 종류

1. 고유 인덱스(UNIQUE INDEX)

- 중복값이 포함될 수 없음
- PRIMARY KEY 제약조건을 생성하면 자동으로 생성됨

2. 비고유 인덱스(NONUNIQUE INDEX)

- 빈번하게 사용되는 일반 컬럼을 대상으로 생성함
- 주로 성능 향상을 위한 목적으로 생성함

3. 단일 인덱스(SINGLE INDEX)

- 한 개의 컬럼으로 구성된 인덱스

4. 결합 인덱스(COMPOSITE INDEX)

- 두 개 이상의 컬럼으로 구성된 인덱스

5. 함수 기반 인덱스(FUNCTION-BASED INDEX)

- SELECT 절이나 WHERE 절에 산술계산식이나 함수식이 사용된 경우
- 계산식은 인덱스의 적용을 받지 않는다.

Oracle Object - INDEX

UNIQUE INDEX

```
CREATE UNIQUE INDEX IDX_EMPNO  
ON EMPLOYEE (EMP_NO);
```

Unique index IDX_EMPNO가 생성되었습니다.

```
SELECT * FROM USER_IND_COLUMNS  
WHERE TABLE_NAME = 'EMPLOYEE';
```

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
1 엔터티1_PK	EMPLOYEE	EMP_ID	1	3	3	ASC
2 IDX_EMPNO	EMPLOYEE	EMP_NO	1	14	14	ASC

```
CREATE UNIQUE INDEX IDX_DEPTCODE  
ON EMPLOYEE (DEPT_CODE);
```

오류 보고 -

SQL 오류: ORA-01452: cannot CREATE UNIQUE INDEX; duplicate keys found
01452. 00000 - "cannot CREATE UNIQUE INDEX; duplicate keys found"

*Cause:

*Action:

**** UNIQUE INDEX는 중복 값이 있는 컬럼에 생성시 에러 발생한다.**

Oracle Object - INDEX

NONUNIQUE INDEX

```
CREATE INDEX IDX_DEPTCODE  
ON EMPLOYEE (DEPT_CODE);
```

| Index IDX_DEPTCODE이 (가) 생성되었습니다.

```
SELECT * FROM USER_IND_COLUMNS  
WHERE TABLE_NAME = 'EMPLOYEE';
```

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
1 엔터티1_PK	EMPLOYEE	EMP_ID	1	3	3	ASC
2 IDX_EMPNO	EMPLOYEE	EMP_NO	1	14	14	ASC
3 IDX_DEPTCODE	EMPLOYEE	DEPT_CODE	1	2	2	ASC

** NONUNIQUE INDEX는 중복 값이 있는 컬럼에도 생성 가능하다.

Oracle Object - INDEX

COMPOSITE INDEX

```
CREATE INDEX IDX_DEPT  
ON DEPARTMENT (DEPT_ID, DEPT_TITLE);
```

| Index IDX_DEPT이 (가) 생성되었습니다.

```
SELECT * FROM USER_IND_COLUMNS  
WHERE TABLE_NAME = 'DEPARTMENT';
```

	INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
1	엔터티1 PK2	DEPARTMENT	DEPT_ID	1	2		2 ASC
2	IDX_DEPT	DEPARTMENT	DEPT_ID	1	2		2 ASC
3	IDX_DEPT	DEPARTMENT	DEPT_TITLE	2	35		35 ASC

** COMPOSITE INDEX는 두 개 이상의 컬럼을 하나의 인덱스로 생성할 수 있다.

** COLUMN_POSITION의 순서에 의해 성능이 차이날 수 있다.

Oracle Object - INDEX

FUNCTION- BASED INDEX

```
CREATE TABLE EMP_SAL
AS SELECT EMP_ID,
        EMP_NAME,
        SALARY,
        BONUS
        (SALARY + (SALARY + NVL(BONUS, 0))) * 12 연봉
FROM EMPLOYEE;
```

Table EMP_SAL이 (가) 생성되었습니다.

```
CREATE INDEX IDX_SALCALC
ON EMP_SAL ((SALARY + (SALARY * NVL(BONUS, 0))) * 12);
```

Index IDX_SALCALC이 (가) 생성되었습니다.

```
SELECT * FROM USER_IND_COLUMNS
WHERE TABLE_NAME = 'EMP_SAL';
```

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
IDX_SALCALC	EMP_SAL	SYS_NC00006\$	1	22	0	ASC

Oracle Object - INDEX

FUNCTION- BASED INDEX

```
CREATE TABLE EMP_SAL
AS SELECT EMP_ID,
        EMP_NAME,
        SALARY,
        BONUS
        (SALARY + (SALARY + NVL(BONUS, 0))) * 12 연봉
FROM EMPLOYEE;
```

Table EMP_SAL이 (가) 생성되었습니다.

```
CREATE INDEX IDX_SALCALC
ON EMP_SAL ((SALARY + (SALARY * NVL(BONUS, 0))) * 12);
```

Index IDX_SALCALC이 (가) 생성되었습니다.

```
SELECT * FROM USER_IND_COLUMNS
WHERE TABLE_NAME = 'EMP_SAL';
```

INDEX_NAME	TABLE_NAME	COLUMN_NAME	COLUMN_POSITION	COLUMN_LENGTH	CHAR_LENGTH	DESCEND
IDX_SALCALC	EMP_SAL	SYS_NC00006\$	1	22	0	ASC

INDEX 재생성

DML작업 (특히 DELETE)명령을 수행한 경우, 해당 인덱스 엔트리가 논리적으로만 제거되고 실제 엔트리는 그냥 남아있게 된다.
제거된 인덱스가 필요 없는 공간을 차지하고 있기 때문에 인덱스를 재생성 할 필요가 있다.

표현식

```
ALTER INDEX 인덱스명 REBUILD;
```

Oracle Object - INDEX

INDEX 활용한 정렬

```
SELECT * FROM EMPLOYEE  
ORDER BY EMP_NO;
```

```
SELECT * FROM EMPLOYEE  
WHERE EMP_NO > '0';
```

SQL | 인출된 모든 행: 24(0초)

EMP_ID	EMP_NAME	EMP_NO	EMAIL	PHONE	DEPT_CODE	JOB_CODE	SAL_LEVEL	SALARY	BONUS	MANAGER_ID	HIRE_DATE	ENT_DATE	ENT_YN
1 200	선동일	621235-1985634	sun_di@kh.or.kr	01099546325	D9	J1	S1	8000000	0.3 (null)		90/02/06	(null)	N
2 201	송종기	631156-1548654	song_jk@kh.or.kr	01045686656	D9	J2	S1	6000000	(null) 200	200	01/09/01	(null)	N
3 202	노홍철	861015-1356452	no_hc@kh.or.kr	01066656263	D9	J2	S4	3700000	(null) 201	201	01/01/01	(null)	N
4 203	송은희	631010-2653546	song_eh@kh.or.kr	01077607879	D6	J4	S5	2800000	(null) 204	204	96/05/03	(null)	N
5 204	유재식	660508-1342154	yoo_js@kh.or.kr	01099999129	D6	J3	S4	3400000	0.2 200	200	00/12/29	(null)	N
6 205	정종하	770102-1357951	jung_jh@kh.or.kr	01036654875	D6	J3	S4	3900000	(null) 204	204	99/09/09	(null)	N
7 206	박나라	630709-2054321	pack_nr@kh.or.kr	01096935222	D5	J7	S6	1800000	(null) 207	207	08/04/02	(null)	N
8 207	하미유	690402-2040612	ha_iy@kh.or.kr	01036654488	D5	J5	S5	2200000	0.1 200	200	94/07/07	(null)	N
9 208	김해솔	870927-1313564	kim_hs@kh.or.kr	01078634444	D5	J5	S5	2500000	(null) 207	207	04/04/30	(null)	N
10 209	심봉선	750206-1325546	sim_bs@kh.or.kr	0113654485	D5	J3	S4	3500000	0.15 207	207	11/11/11	(null)	N
11 210	윤은혜	650505-2356985	youn_eh@kh.or.kr	0179964233	D5	J7	S5	2000000	(null) 207	207	01/02/03	(null)	N
12 211	전철돈	830807-1121321	jun_hd@kh.or.kr	01044432222	D8	J6	S5	2000000	(null) 200	200	12/12/12	(null)	N
13 212	장프위	780923-2234542	jang_zw@kh.or.kr	01066682224	D8	J6	S5	2550000	0.25 211	211	15/06/17	(null)	N
14 213	하동운	621111-1785463	ha_dh@kh.or.kr	01158456632	(null)	J6	S5	2320000	0.1 (null)	(null)	99/12/31	(null)	N
15 214	방명수	856795-1313513	bang_ms@kh.or.kr	01074127545	D1	J7	S6	1380000	(null) 200	200	10/04/04	(null)	N
16 215	대복존	881130-1050911	dae_bh@kh.or.kr	01088808584	D5	J5	S4	3760000	(null) (null)	(null)	17/06/19	(null)	N
17 216	차태연	770808-1364897	cha_ty@kh.or.kr	01064643212	D1	J6	S5	2780000	0.2 214	214	13/03/01	(null)	N
18 217	전지연	770808-2665412	jun_jy@kh.or.kr	01033624442	D1	J6	S4	3660000	0.3 214	214	07/03/20	(null)	N
19 218	미오리	870427-2232123	loo_or@kh.or.kr	01022306545	(null)	J7	S5	2890000	(null) (null)	(null)	16/11/28	(null)	N
20 219	임시환	660712-1212123	im_sw@kh.or.kr	(null)	D2	J4	S6	1550000	(null) (null)	(null)	99/09/09	(null)	N
21 220	이종석	770823-1113111	lee_js@kh.or.kr	(null)	D2	J4	S5	2490000	(null) (null)	(null)	14/09/18	(null)	N
22 221	유하진	800808-1123341	yoo_hj@kh.or.kr	(null)	D2	J4	S5	2480000	(null) (null)	(null)	94/01/20	(null)	N
23 222	이태림	760918-2854697	lee_tr@kh.or.kr	01033000002	D8	J6	S5	2436240	0.35 100	100	97/09/12	17/09/12	Y
24 900	장채현	901123-1080503	jang_ch@kh.or.kr	01055569512	D1	J8	S3	4300000	0.2 200	200	17/09/19	(null)	N

** 둘 다 주민번호 기준 오름차순 정렬이지만, ORDER BY로 정렬하는 것 보다
인덱스를 활용하는 것이 더 좋은 성능을 보인다.

ORACLE OBJECT(SYNONYM)

SYNONYM이란?

사용자가 다른 사용자의 객체를 참조할 때 [사용자ID].[테이블명]으로 표시해야 한다.
이처럼 길게 표현되는 것을 동의어(SYNONYM)로 설정하고 간단히 사용할 수 있다.

1. 비공개 동의어

- 객체에 대한 접근 권한을 부여받은 사용자가 정의한 동의어
- 해당 사용자만 사용할 수 있음

2. 공개 동의어

- 권한을 주는 사용자(DBA)가 정의한 동의어
- 모든 사용자가 사용할 수 있음(PUBLIC)

예) DUAL

Oracle Object - SYNONYM

SYNONYM 생성

```
CREATE SYNONYM EMP FOR EMPLOYEE;
```

오류 보고 -

SQL 오류: ORA-01031: insufficient privileges

01031. 00000 - "insufficient privileges"

*Cause: An attempt was made to perform a database operation without the necessary privileges.

*Action: Ask your database administrator or designated security administrator to grant you the necessary privileges

SYSTEM 계정

```
GRANT CREATE SYNONYM TO EMPLOYEE;
```

Grant를 (를) 성공했습니다.




EMPLOYEE 계정




```
CREATE SYNONYM EMP FOR EMPLOYEE;
```

Synonym EMP이 (가) 생성되었습니다.

```
SELECT * FROM EMPLOYEE;
```

```
SELECT * FROM EMP;
```

    SQL | 인출된 모든 행: 24(0초)

    SQL | 인출된 모든 행: 24(0초)

Oracle Object - SYNONYM

SYNONYM 생성

SYSTEM 계정

```
CREATE PUBLIC SYNONYM DEPT FOR EMPLOYEE.DEPARTMENT;
```

Public synonym DEPT이 (가) 생성되었습니다.

SYSTEM 계정

```
SELECT * FROM EMPLOYEE.DEPARTMENT;
```

```
SELECT * FROM DEPT;
```

EMPLOYEE 계정

```
SELECT * FROM DEPARTMENT;
```

```
SELECT * FROM DEPT;
```

SQL | 인출된 모든 행: 9(0초)

	DEPT_ID	DEPT_TITLE	LOCATION_ID
1	D1	인사관리부	L1
2	D2	회계관리부	L1
3	D3	마케팅부	L1
4	D4	국내영업부	L1
5	D5	해외영업1부	L2
6	D6	해외영업2부	L3
7	D7	해외영업3부	L4
8	D8	기술지원부	L5
9	D9	총무부	L1

Oracle Object - SYNONYM

SYNONYM 삭제

SYSTEM 계정

```
DROP PUBLIC SYNONYM DEPT;
```

Public synonym DEPT이 (가) 삭제되었습니다.

EMPLOYEE 계정

```
DROP SYNONYM EMP;
```

Synonym EMP이 (가) 삭제되었습니다.

ORACLE OBJECT (PROCEDURE & FUNCTION)

Stored Procedure이란?

PL/SQL문을 저장하는 객체이다. 필요할 때마다 복잡한 구문을 다시 입력 할 필요 없이 간단하게 호출해서 실행결과를 얻을 수 있다.

Procedure의 특징

프로시저는 일련의 작업 절차를 정리해서 저장해 둔 것
여러 SQL문을 묶어서 미리 정의해 두고 하나의 요청으로 실행 할 수 있음
자주 사용되는 복잡한 작업들을 간단하게 미리 만들어 두면 쉽게 사용이 가능함
저장 프로시저를 사용하면 성능도 향상 될 수 있음

Procedure 작성 및 실행

매개변수의 데이터형의 크기는 지정하면 안됨

SELECT 사용시 INTO를 통하여 변수에 값 저장 해야 함

CREATE PROCEDURE 프로시저이름 (매개변수명1 [IN|OUT|INOUT] 자료형, 매개변수명2[MODE] 자료형...)

-- IN : 데이터를 전달 받을 때

-- OUT : 수행된 결과를 받아갈 때

-- INOUT : 두 가지 목적에 모두 사용 (실제로는 사용되지 않음!)

IS

지역변수 선언;

BEGIN

실행할 문장 1;

실행할 문장 2;

실행할 문장 3;

-- 함수호출, 초기화, 절차적 요소, SQL 문

END;

/

Oracle - Procedure&Function

Procedure 작성 및 실행

```
EXECUTE 프로시저[(전달값1, 전달값2, ...)];
```

Procedure 확인

```
SELECT * FROM ALL_PROCEDES; -> 모든 프로시저 확인
```

```
SELECT * FROM USER_PROCEDES; -> 유저가 생성한 프로시저 목록들 확인
```

```
SELECT * FROM USER_SOURCE WHERE NAME = '프로시저 이름; -> 생성된 프로시저 코드 확인
```

Procedure 삭제

```
DROP PROCEDURE 프로시저이름;
```

Oracle - Procedure&Function

Procedure 예제

```
CREATE TABLE EMP_DUP  
AS SELECT * FROM EMPLOYEE;
```

Table EMP_DUP이 (가) 생성되었습니다.

```
CREATE OR REPLACE PROCEDURE  
DEL_ALL_EMP  
IS  
BEGIN  
  DELETE FROM EMP_DUP;  
  COMMIT;  
END;  
/
```

Procedure DEL_ALL_EMP이 (가) 컴파일되었습니다.



```
SELECT * FROM EMP_DUP;
```

SQL | 인출된 모든 행: 24(0.015초)

```
EXEC DEL_ALL_EMP;
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.



```
SELECT * FROM EMP_DUP;
```

SQL | 인출된 모든 행: 0(0초)

Oracle - Procedure&Function

Procedure 확인

```
SELECT * FROM ALL_PROCEDURES;
```

-> 모든 프로시저 확인

```
SELECT * FROM USER_PROCEDURES;
```

-> 사용자가 생성한 프로시저 목록들을 확인

```
SELECT * FROM USER_SOURCE WHERE NAME = '프로시저 이름;
```

-- 생성된 프로시저 코드를 볼 수 있음

ex)

```
SELECT * FROM USER_SOURCE WHERE NAME = 'DEL_ALL_EMP';
```

Oracle - Procedure&Function

IN 모드 , OUT 모드, 바인드 변수

IN : 데이터를 전달 받을 때, OUT : 수행된 결과를 받아갈 때,
바인드 변수 : 값을 받아 사용하는 변수

```
CREATE OR REPLACE PROCEDURE SELECT_EMPID(  
  V_EMP_ID IN EMPLOYEE.EMP_NO%TYPE,           -- IN 모드 매개변수  
  V_EMP_NAME OUT EMPLOYEE.EMP_NAME%TYPE,       -- OUT 모드 매개변수  
  V_SALARY OUT EMPLOYEE.SALARY%TYPE,           -- OUT 모드 매개변수  
  V_BONUS OUT EMPLOYEE.BONUS%TYPE             -- OUT 모드 매개변수  
)  
IS  
BEGIN  
  SELECT EMP_NAME, SALARY, NVL(BONUS, 0)  
  INTO V_EMP_NAME, V_SALARY, V_BONUS  
  FROM EMPLOYEE  
  WHERE EMP_ID = V_EMP_ID;  
END;  
/
```


Oracle - Procedure&Function

IN 모드 , OUT 모드, 바인드 변수

IN : 데이터를 전달 받을 때, OUT : 수행된 결과를 받아갈 때,

바인드 변수 : 값을 받아 사용하는 변수

```
VARIABLE VAR_EMP_NAME VARCHAR2(30);      -- 바인드 변수
VARIABLE VAR_SALARY NUMBER;                -- 바인드변수와 매개변수의 자료형은
반드시 같아야 함.
VARIABLE VAR_BONUS NUMBER;

EXEC SELECT_EMPID(200, :VAR_EMP_NAME, :VAR_SALARY, :VAR_BONUS);
-> 바인드 변수 사용

PRINT VAR_EMP_NAME;
PRINT VAR_SALARY;
PRINT VAR_BONUS;
```

Oracle - Procedure&Function

IN 모드 , OUT 모드, 바인드 변수

IN : 데이터를 전달 받을 때, OUT : 수행된 결과를 받아갈 때,
바인드 변수 : 값을 받아 사용하는 변수

생성된 프로시저 확인

```
SELECT * FROM USER_PROCEDURES;
```

-> 유저가 생성한 프로시저 목록들을 확인할 수 있음

```
SELECT * FROM USER_SOURCE WHERE NAME = '프로시저 이름;
```

-> 생성된 프로시저 코드를 볼 수 있음

Function이란?

- 프로시저와 거의 유사한 용도로 사용하지만, 실행 결과를 되돌려 받을 수 있는 점에서 프로시저와 다름,
- 리턴값이 반드시 존재하는 객체

저장 함수 작성 및 실행 방법

```
CREATE [OR REPLACE] FUNCTION 함수이름(매개변수명 자료형,...)
RETURN 자료형;
IS
    지역변수선언;
BEGIN
    실행할 SQL문;
    RETURN 데이터;
END;
/
```

생성된 함수 확인 및 삭제

```
SELECT * FROM USER_PROCEDURES;
```

-> 유저가 생성한 프로시저 목록들을 확인(함수도 하나의 프로시저)

```
SELECT * FROM USER_SOURCE WHERE NAME = '함수 이름;
```

-> 생성된 프로시저 코드를 볼 수 있음

```
DROP FUNCTION BONUS_CALC;
```

-> 프로시저 함수 삭제

Function 실습 예제

사번을 전달받아 사원에게 보너스를 지급하려고 하는데,급여의 200%에 해당하는 비용을 지급하려 한다. 그 금액을 FUNCTION을 통해 리턴 받아 출력하시오

저장함수명 : BONUS_CAL

```
CREATE FUNCTION BONUS_CAL(V_ID IN EMPLOYEE.EMP_ID%TYPE)
RETURN NUMBER
IS
    VSALARY EMPLOYEE.SALARY%TYPE;
    VBONUS NUMBER;
BEGIN
    SELECT SALARY
    INTO VSALARY
    FROM EMPLOYEE
    WHERE EMP_ID = V_ID;
    VBONUS := VSALARY * 2;
    RETURN VBONUS;
END;
/
```

```
VARIABLE RESULT NUMBER;
EXEC :RESULT := BONUS_CAL('&EMPNO');
```

ORACLE OBJECT (CURSOR)

Cursor(커서)란?

- 질의의 결과로 얻어진 여러 행이 저장된 메모리상의 위치
- SELECT 문의 수행 결과가 여러개의 ROW(행)로 구해지는 경우에 모든 행에 대해 어떠한 처리를 하고 싶을 때 사용 하는 객체
- CURSOR ~ OPEN ~ FETCH ~ CLOSE 로 된 4단계로 진행

CURSOR의 상태

속 성	설 명
%NOTFOUND	커서 영역의 자료가 모두 FETCH되어 다음 영역이 존재하지 않으면 TRUE
%FOUND	커서 영역에 아직 FETCH되지 않은 자료가 있으면 TRUE
%ISOPEN	커서가 OPEN된 상태이면 TRUE
%ROWCOUNT	커서가 얻어 온 레코드의 갯수

암시적 커서

암시적 커서는 사용자가 직접 쿼리의 결과에 접근하지 못함 자동적으로 해줌

암시적 커서 예제

```
CREATE TABLE EMPLOYEE2 AS SELECT * FROM EMPLOYEE;
```

```
CREATE PROCEDURE Test11                                -- 프로시저 생성
IS
v_conut_row NUMBER;
BEGIN
UPDATE EMPLOYEE2                                     -- 업데이트 진행
SET EMP_NAME='홍길동'
WHERE EMP_ID = 200;
v_conut_row := SQL%ROWCOUNT;                        -- 성공된 행의 갯수가 ROWCOUNT에 저장됨
DBMS_OUTPUT.PUT_LINE('성공 행 갯수 : ' || v_conut_row); -- 갯수 출력
END;
/
EXEC Test11;                                           -- 프로시저 실행
```


명시적 커서

- 사용자가 직접 쿼리의 결과에 접근해서 이를 사용하기 위해 명시적으로 선언한 커서
- 기존 PL/SQL 이나 프로시저에서는 하나의 결과만 나와야 했으나 명시적 커서를 이용하면 여러 개에 대한 처리가 가능

명시적 커서의 형식

(4단계 : 선언→ 열기→ 처리→ 닫기)

```
DECLARE
    CURSOR cursor_name          -- 커서 선언
    IS statement;               -- 커서에 사용된 SQL 구문
BEGIN
    OPEN cursor_name;           -- 커서 열기
    FETCH cur_name INTO variable_name; -- 커서로부터 데이터를 읽어와 변수에 저장
    CLOSE cursor_name;          -- 커서 닫기
END;
```

명시적 커서

1 단계 : 커서 선언

- > 암시적 커서와 달리 사용할 커서에 이름을 부여하고 이 커서에 대한 쿼리를 선언 해야함
- > 명시적 커서란 것이 결과 데이터 집합을 로우별로 참조해서 무언가를 작업하기 위한 용도 이므로 당연히 커서를 정의해 사용하는 쿼리문은 SELECT 문이어야 함

```
CURSOR 커서명(매개변수1,매개변수2,...)  
IS  
SELECT 문장;
```

2 단계 : 커서 열기

- > 커서를 선언한 뒤 해당 커서를 사용하려면 커서를 열어야 함

```
OPEN 커서명 (매개변수1,매개변수2,...);
```

명시적 커서

3 단계 : 패치 단계에서 커서 사용

- > 정의한 커서를 열고 난 후에야 SELECT 문의 결과로 반환되는 로우에 접근할 수 있음
- > 결과 집합의 로우 수는 보통 1개 이상이므로 개별 로우에 접근하기 위해서는 반복문을 사용할 수 있음
 - LOOP, WHILE, FOR 문 사용이 가능
- > 반복을 하게 되면 자동으로 다음 로우를 가리키게 됨

LOOP

```
    FETCH 커서명 INTO 변수1, 변수2, ...;  
    EXIT WHEN 커서명%NOTFOUND;    -- 커서의 속성을 이용하여 루프를 벗어남  
END LOOP;
```

4 단계 : 커서 닫기

- > 패치 작업이 끝나고 반복문을 빠져 나오면 커서 사용이 모두 끝났으므로 반드시 커서를 닫아 주어야 함
 - 우리가 스트림을 열고 닫는 개념과 같음

명시적 커서 - 예제

```
DECLARE
    USER    EMPLOYEE2%ROWTYPE;
    CURSOR testCursor (DC EMPLOYEE2.DEPT_CODE%TYPE)  -- 커서 선언
    IS SELECT * FROM EMPLOYEE2 WHERE DEPT_CODE = DC;
    -- 커서에 사용된 SQL 구문
BEGIN
    OPEN testCursor('D5');          -- 커서 열기
    LOOP
        FETCH testCursor INTO USER;  -- 커서로부터 데이터를 읽어와 변수에 저장
        EXIT WHEN testCursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('사번 : ' || USER.EMP_ID);
        DBMS_OUTPUT.PUT_LINE('사원명 : ' || USER.EMP_NAME);
        DBMS_OUTPUT.PUT_LINE('부서코드 : ' || USER.DEPT_CODE);
    END LOOP;
    CLOSE testCursor;                -- 커서 닫기
END;
/
```

ORACLE OBJECT (PACKAGE)

Oracle - Package

Package(패키지)란?

- 자바의 패키지과 동일한 개념으로 같은 테이블을 참조하는 프로시저나 함수를 같은 패키지로 묶을 수 있음

패키지 선언 방법

```
CREATE [OR REPLACE] PACKAGE 패키지명  
IS  
    정의될 저장프로시저와 저장함수  
END;  
/
```

패키지에 소속된 프로시저나 함수 실행

```
EXEC 패키지명.저장프로시저명 [(전달값,...)];
```

패키지(Package) 형식

```
CREATE [OR REPLACE] PACKAGE package_name  
  
IS  
    PROCEDURE procedure_name1;  
    PROCEDURE procedure_name2;  
  
END;  
/
```

```
CREATE [OR REPLACE] PACKAGE package_name  
IS  
    PROCEDURE procedure_name1  
    IS  
        ...  
    END;  
  
END;  
/
```

ORACLE OBJECT (TRIGGER)

Trigger(트리거)란?

- 데이터베이스가 미리 정해놓은 조건을 만족하거나 어떠한 동작이 수행되면 자동적으로 수행되는 행동을 말함
- 트리거는 테이블이나 뷰가 INSERT, UPDATE, DELETE등의 DML문에 의해 데이터가 입력,수정,삭제될 경우 자동으로 실행

트리거의 실행 시점

트리거 실행 시점을 이벤트 전(BEFORE)이나 이벤트 후(AFTER)로 지정하여 설정

트리거의 이벤트

트리거의 이벤트는 사용자가 어떤 DML(INSERT, UPDATE, DELETE)문을 실행했을 때 트리거를 발생시킬 것인지를 결정

트리거의 몸체

트리거의 몸체는 해당 타이밍에 해당 이벤트가 발생했을 때 실행될 기본 로직이 포함되는 부분으로 BEGIN ~ END에 안에 작성함

트리거의 유형

- 트리거의 유형은 FOR EACH ROW에 의해 문장 레벨 트리거와 행 레벨 트리거로 나뉨
- FOR EACH ROW가 생략되면 "문장 레벨 트리거"
- FOR EACH ROW가 정의되면 "행 레벨 트리거"
- 문장 레벨 트리거는 어떤 사용자가 트리거가 설정되어 있는 테이블에 대해 DML(INSERT, UPDATE, DELETE)문을 실행시킬 때 트리거를 단 한번 발생 시킴
- 행 레벨 트리거는 DML(INSERT, UPDATE, DELETE)에 의해서 여러 개의 행이 변경된다면 각 행이 변경될때마다 트리거를 발생시키는 방법
(만약 5개의 행이 변경되면 5번의 트리거가 발생함)

트리거의 조건

트리거의 조건은 행 레벨 트리거에서만 설정할 수 있으며 트리거 이벤트에 정의된 테이블에 이벤트가 발생할 때보다 구체적인 데이터 검색 조건을 부여할 때 사용됨

Oracle - Trigger

트리거 확인하기

```
SELECT * FROM ALL_TRIGGERS;
```

```
SELECT * FROM USER_TRIGGERS;
```

트리거 예제 1

회원이 탈퇴 하면 탈퇴 테이블에 정보를 옮겨 놓는 PL/SQL
트리거 실습을 위한 테이블 2개 생성 (회원 테이블 , 탈퇴 회원 테이블)

트리거 예제 2

회원이 탈퇴 하면 탈퇴 테이블에 정보를 옮겨 놓는 트리거 설정

Oracle - Trigger

바인드 변수

:NEW - 새로 입력된 (INSERT, UPDATE) 데이터

:OLD - 기존 데이터

(FOR EACH ROW를 사용)

:OLD.컬럼명 -> SQL 반영 전의 컬럼 데이터

:NEW.컬럼명 -> SQL 반영 후의 컬럼 데이터

※ DELETE에는 삭제 되기 때문에 OLD만 사용할 수 있음

바인드 변수 연습 예제

회원 가입시 메시지가 뜨는 트리거 설정 해보기

ORACLE OBJECT (ROLE)

Oracle - Role

Role이란? Connect,Resource

사용자에게 여러개의 권한을 한번에 부여 할 수 있는 개체

사용자에게 권한을 부여할때 한개씩 부여하게 된다면 권한 부여 및 회수에 따른 관리가 불편함

Oracle DB 설치시 기본 제공되는 ROLE

CONNECT : 사용자가 데이터 베이스에 접속 가능하도록 하기 위한 권한이 있는 Role

RESOURCE : 사용자가 객체(테이블,뷰,인덱스)를 생성하기 위한 시스템 권한 제공되는 Role

DBA : 시스템 자원을 무제한적으로 사용가능하며 시스템 관리를 하기 위한 모든 권한 Role

Oracle - Role

사용자 Role 생성 / 부여 실습

emptyRole : 아무런 권한이 포함되지 않은 Role
connectRole : 접속 권한만 있는 Role
managerRole : 접속 권한 + Table 생성 권한이 있는 Role

SYSDBA 계정으로 Role 생성

```
CREATE ROLE emptyRole;  
CREATE ROLE connectRole;  
CREATE ROLE managerRole;
```

SYSDBA 계정으로 생성된 Role에 권한 부여

```
GRANT CREATE SESSION TO connectRole;  
GRANT CREATE SESSION,CREATE TABLE TO managerRole;
```

사용자 Role 생성 / 부여 실습

```
CREATE USER testkh IDENTIFIED BY testkh;    -- ID : testkh, PW : testkh 생성  
GRANT emptyRole TO testkh;                -- 권한이 없는 emptyRole 을 testkh에 부여
```

그 후 SQL Developer 에서 testkh 계정으로 새로운 접속을 만들어서 접속 시도 해보기 (실패)

- 접속 이름 : web_testkh
- 사용자 이름 : testkh
- 비밀번호 : testkh

```
REVOKE emptyRole FROM testkh;    -- emptyRole 권한 회수 (SYSDBA에서)
```

```
GRANT connectRole TO testkh;    -- 접속권한만 있는 connectRole을 testkh에 부여  
ALTER USER testkh QUOTA UNLIMITED ON SYSTEM;  
-- 테이블을 생성하려면 테이블 스페이스의 권한의 필요함
```

다시 SQL Developer 에서 testkh 계정으로 새로운 접속을 만들어서 접속 시도 해보기 (접속 성공 할것임)

- 접속 이름 : web_testkh
- 사용자 이름 : testkh
- 비밀번호 : testkh

사용자 Role 생성 / 부여 실습

접속된 testkh 계정으로 테이블 생성 시도 해보기 (실패 - 권한이 없음)

- CREATE TABLE testTb(NO NUMBER, NAME VARCHAR2(20));
- insufficient privileges 발생 (불충분한 권한)

REVOKE connectRole FROM testkh;

-- connectRole 권한 회수(SYSDBA에서)

GRANT managerRole TO testkh;

-- 접속권한,테이블 생성 권한이 있는 managerRole을 testkh에 부여

접속된 testkh 계정으로 테이블 생성 시도 해보기 (성공)

- CREATE TABLE testTb(NO NUMBER, NAME VARCHAR2(20));