

# Ch 5.

---

## DDL(Data Definition Language)

5.1 | 테이블

5.2 | 뷰<sup>View</sup>

5.3 | 시퀀스<sup>Sequence</sup>

5.4 | 인덱스<sup>Index</sup>

# Objective

- ❖ 데이터를 저장하는 테이블을 생성할 수 있다
- ❖ 테이블 구조를 변경할 수 있다
- ❖ 제약조건<sup>Constraints</sup>을 사용하여 데이터 무결성<sup>Data Integrity</sup>을 보장할 수 있다
- ❖ 뷰를 생성하고 활용할 수 있다
- ❖ 시퀀스를 생성하고 활용할 수 있다
- ❖ 인덱스 개념을 이해하고 쿼리 성능 향상을 위해 인덱스를 활용할 수 있다

# Ch 5.

## DDL(Data Definition Language)

5.1 | 테이블

5.2 | 뷰 View

5.3 | 시퀀스 Sequence

5.4 | 인덱스 Index

## 5.1.1 테이블 생성

[기본 구문]

```
CREATE TABLE table_name  
( column_name datatype [DEFAULT expr] [ column_constraint ] [, ... ]  
[ , table_constraint , ... ] ) ;
```

```
column_constraint  
[ CONSTRAINT constraint_name ] constraint_type  
table_constraint  
[ CONSTRAINT constraint_name ] constraint_type (column_name , ...)
```

[구문 설명]

- **table\_name, column\_name**  
테이블 이름 지정, 컬럼 이름 지정
- **datatype**  
컬럼의 데이터 타입, 크기 지정
- **DEFAULT expr**  
해당 컬럼에 적용될 자동 기본 값
- **CONSTRAINTS**
  - COLUMN\_CONSTRAINT : 컬럼 레벨에서의 제약 조건
  - TABLE\_CONSTRAINT : 테이블 레벨에서의 제약 조건

### 5.1.1 테이블 생성 - Naming Rule

- 테이블 및 컬럼 이름
  - 문자로 시작, 30자 이하
  - 영문 대/소문자(A ~ Z, a ~ z), 숫자(0 ~ 9), 특수문자( \_, \$, # ), 한글만 포함 가능
- 중복되는 이름은 사용할 수 없음
- 예약 키워드(CREATE, TABLE, COLUMN 등)는 사용할 수 없음

## 5.1.1 테이블 생성 예

```
CREATE TABLE TEST  
(ID      NUMBER(5),  
 NAME    CHAR(10),  
 ADDRESS VARCHAR2(50)  
);
```

☞ 테이블 이름 : TEST

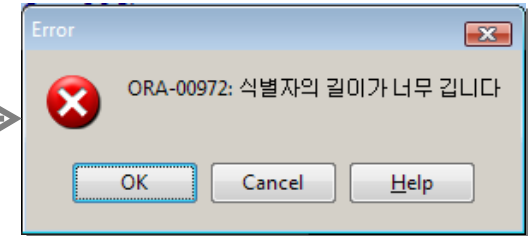
☞ 컬럼 정의 : 이름 - ID, 타입 - NUMBER

☞ 컬럼 정의 : 이름 - NAME, 타입 - CHAR(10)

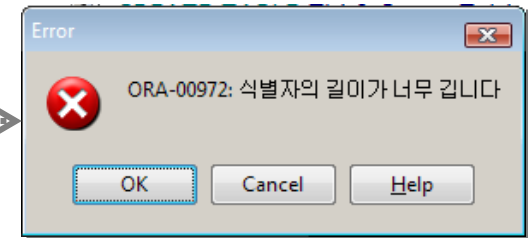
☞ 컬럼 정의 : 이름 - ADDRESS, 타입 - VARCHAR2(50)

## 5.1.1 테이블 생성 - 생성 오류 예

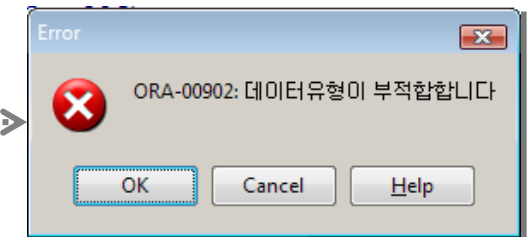
```
CREATE TABLE THISISCREATETABLESAMPLEOVER30CHARACTERS  
(COL1 CHAR(10));
```



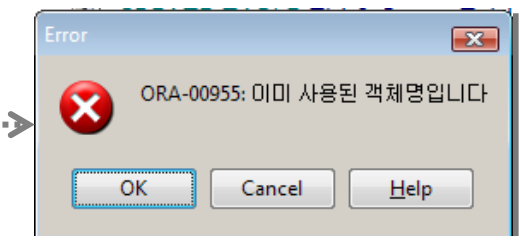
```
CREATE TABLE SAMPLETABLE  
(THISISSAMPLECOLUMNOVER30CHARACTER CHAR(10));
```



```
CREATE TABLE SAMPLETABLE  
(COL-1 CHAR(5));
```



```
CREATE TABLE TEST  
(COL1 CHAR(10));
```



## 5.1.1 테이블 생성 - 실습

테이블 이름 : ORDERS

컬럼이름	ORDERNO	CUSTNO	ORDERDATE	SHIPDATE	SHIPADDRESS	QUANTITY
설명	주문번호	고객번호	주문일자	배송일자	배송주소	주문수량
데이터타입	CHAR(4)	CHAR(4)	DATE	DATE	VARCHAR2(40)	NUMBER
DEFAULT	-	-	SYSDATE	-	-	-

[생성 구문]

```
CREATE TABLE ORDERS
(ORDERNO      CHAR(4),
 CUSTNO       CHAR(4),
 ORDERDATE    DATE DEFAULT SYSDATE,
 SHIPDATE     DATE,
 SHIPADDRESS  VARCHAR2(40),
 QUANTITY     NUMBER);
```



## 5.1.2 제약조건 Constraints

- 데이터 무결성

데이터베이스에 저장되어 있는 데이터가 손상되거나 원래의 의미를 잃지 않고 유지하는 상태

- 데이터 무결성 제약조건

데이터 무결성을 보장하기 위해 오라클에서 지원하는 방법

예) 유효하지 않은 데이터 입력 방지, 유효한 범위에서만 데이터 변경/삭제 작업 허용

제약조건	설명	설정 레벨
NOT NULL	해당 컬럼에 NULL을 포함되지 않도록 함	컬럼
UNIQUE	해당 컬럼 또는 컬럼 조합 값이 유일하도록 함	컬럼, 테이블
PRIMARY KEY	각 행을 유일하게 식별할 수 있도록 함	컬럼, 테이블
REFERENCES <i>TABLE (column_name)</i>	해당 컬럼이 참조하고 있는 테이블 <sup>부모</sup> 테이블의 특정 컬럼 값들과 일치하거나 또는 NULL이 되도록 보장함	컬럼, 테이블
CHECK	해당 컬럼에 특정 조건을 항상 만족시키도록 함	컬럼, 테이블

### 5.1.2 제약조건

- 이름으로 관리
  - 문자로 시작, 길이는 30자까지 가능
  - 이름을 따로 지정하지 않으면 자동 생성 (SYS\_Cxxxxxxx 형식)
- 생성 시기
  - 테이블 생성과 동시
  - 테이블을 생성한 후
- 컬럼 레벨 또는 테이블 레벨에서 정의할 수 있다
  - NOT NULL은 '컬럼 레벨' 에서만 가능
  - 컬럼 여러 개를 조합하는 경우에는 '테이블 레벨' 에서만 가능

## 5.1.2 제약조건 - NOT NULL 사용 예

```
CREATE TABLE TABLE_NOTNULL
(ID    CHAR(3)  NOT NULL,
 SNAME VARCHAR2(20));
```

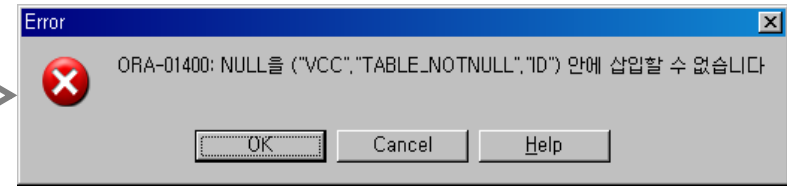
```
INSERT INTO TABLE_NOTNULL
VALUES ('100','ORACLE');
```

```
INSERT INTO TABLE_NOTNULL
VALUES (NULL,'ORACLE');
```

'ID' 컬럼에 NULL을 입력하려고 했기 때문에 발생

Columns of TABLE_NOTNULL		
Name	Type	Nullable
ID	CHAR(3)	
SNAME	VARCHAR2(20)	Y

ID	SNAME
100	ORACLE

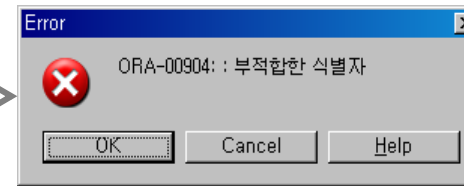


에러 메시지 표시 형식

"VCC"."TABLE\_NOTNULL"."ID" → 계정.테이블.컬럼

```
CREATE TABLE TABLE_NOTNULL2
(ID    CHAR(3),
 SNAME VARCHAR2(20),
 CONSTRAINT TN2_ID_NN NOT NULL (ID));
```

'NOT NULL' 제약조건은 컬럼 레벨에서만 정의 가능



## 5.1.2 제약조건 - UNIQUE : 단일 컬럼 생성 예

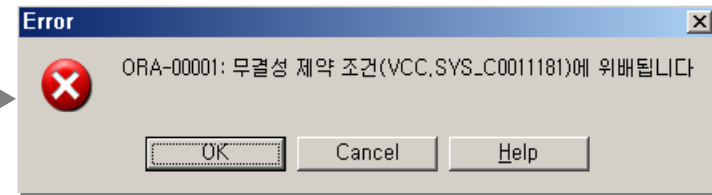
```
CREATE TABLE TABLE_UNIQUE  
(ID    CHAR(3)  UNIQUE,  
 SNAME VARCHAR2(20));
```

```
INSERT INTO TABLE_UNIQUE  
VALUES ('100', 'ORACLE');
```

ID	SNAME
100	ORACLE

```
INSERT INTO TABLE_UNIQUE  
VALUES ('100', 'ORACLE');
```

'ID' 컬럼에 중복 값을 입력하려고  
했기 때문에 발생



제약조건 이름을 지정하지 않았으므로  
임의 이름 "SYS\_C0011181"이 할당됨

## 5.1.2 제약조건 - UNIQUE : 조합 컬럼 생성 예

```
CREATE TABLE TABLE_UNIQUE2
(ID      CHAR(3),
 SNAME  VARCHAR2(20),
 SCODE  CHAR(2),
 CONSTRAINT TN2_ID_UN UNIQUE (ID,SNAME));
```

```
INSERT INTO TABLE_UNIQUE2
VALUES ('100', 'ORACLE', '01');
```

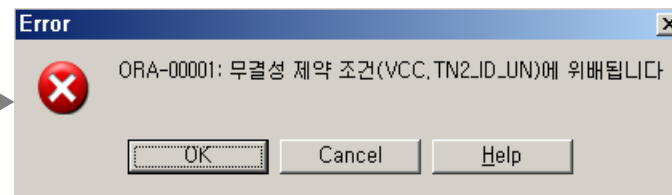
```
INSERT INTO TABLE_UNIQUE2
VALUES ('200', 'ORACLE', '01');
```

```
INSERT INTO TABLE_UNIQUE2
VALUES ('200', 'ORACLE', '02');
```

☞ 컬럼 조합 결과를 유일하게 하려는  
목적이므로 테이블 레벨에서 생성해야 함

ID	SNAME	SCODE
100	ORACLE	01

ID	SNAME	SCODE
100	ORACLE	01
200	ORACLE	01



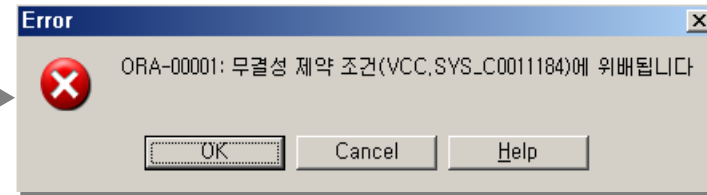
## 5.1.2 제약조건 - UNIQUE : 생성 예

```
CREATE TABLE TABLE_UNIQUE3  
(ID      CHAR(3) UNIQUE,  
 SNAME  VARCHAR2(20) UNIQUE,  
 SCORE  CHAR(2));
```

```
INSERT INTO TABLE_UNIQUE3  
VALUES ('100', 'ORACLE', '01');
```

```
INSERT INTO TABLE_UNIQUE3  
VALUES ('200', 'ORACLE', '01');
```

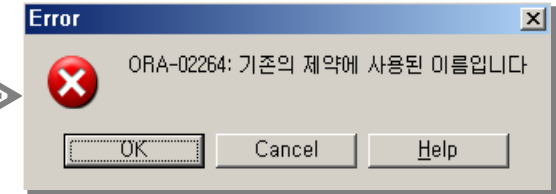
ID	SNAME	SCORE
100	ORACLE	01



'ID' 컬럼과 'SNAME' 컬럼에 각각 설정되었기 때문에, 중복된 'SNAME' 컬럼 값이 입력될 수 없음  
→ 두 컬럼의 조합 결과를 유일하게 하려면 '테이블 레벨'에서 생성

## 5.1.2 제약조건 - UNIQUE : 생성 예

```
CREATE TABLE TABLE_UNIQUE4  
(ID      CHAR(3) CONSTRAINT TN4_ID_UN UNIQUE ,  
 SNAME  VARCHAR2(20) CONSTRAINT TN4_ID_UN UNIQUE ,  
 SCORE  CHAR(2));
```



제약 조건 이름을 동일하게 해서 'ID' 컬럼과 'SNAME' 컬럼 조합 결과를 유일하게 하려고 했음  
→ 두 컬럼의 조합 결과를 유일하게 하려면 '테이블 레벨'에서 생성

## 5.1.2 제약조건 - UNIQUE : 생성 예

UNIQUE 제약 조건이 생성된 컬럼에는 NULL 포함 가능

[단일 컬럼 경우 예]

```
CREATE TABLE TABLE_UNIQUE5
(ID NUMBER UNIQUE,
NAME VARCHAR2(10));
```

ID	NAME
	ORACLE
	SQL
	JAVA

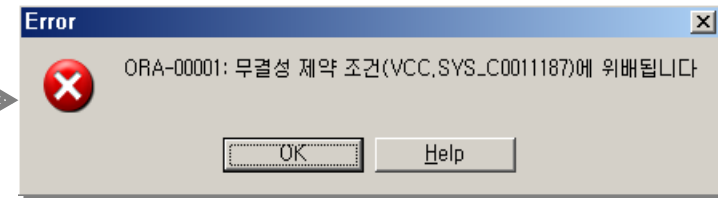
'ID' 컬럼에 NULL 입력 가능

[컬럼 조합 경우 예]

```
CREATE TABLE TABLE_UNIQUE6
(ID NUMBER,
NAME VARCHAR2(10),
UNIQUE (ID, NAME));
```

ID	NAME
	ORACLE
100	SQL
200	JAVA
	ORACLE

'ID, NAME' 컬럼 조합으로 NULL 가능



'ID, NAME' 컬럼 조합 중복



## 5.1.2 제약조건 - PRIMARY KEY

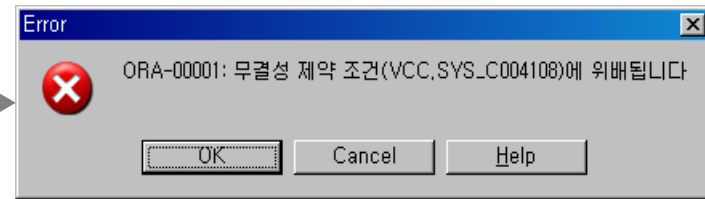
- UNIQUE + NOT NULL 의미
- 테이블 당 1개만 생성 가능

```
CREATE TABLE TABLE_PK  
(ID CHAR(3) PRIMARY KEY,  
SNAME VARCHAR2(20));
```

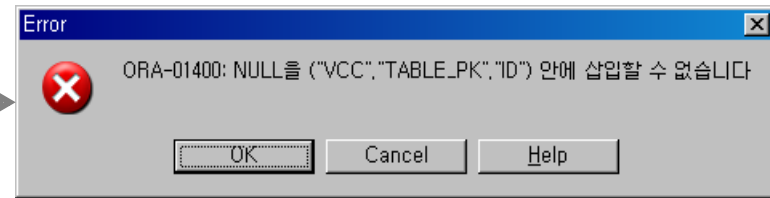
```
INSERT INTO TABLE_PK  
VALUES ('100', 'ORACLE');
```

ID	SNAME
100	ORACLE

```
INSERT INTO TABLE_PK  
VALUES ('100', 'IBM');
```



```
INSERT INTO TABLE_PK  
VALUES (NULL, 'SUN');
```



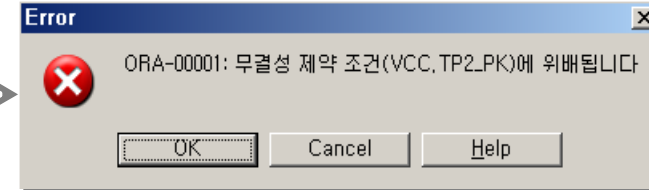
## 5.1.2 제약조건 - PRIMARY KEY : 조합 컬럼 생성 예

```
CREATE TABLE TABLE_PK2
(ID      CHAR(3),
 SNAME  VARCHAR2(20),
 SCORE  CHAR(2),
 CONSTRAINT TP2_PK PRIMARY KEY (ID,SNAME));
```

ID	SNAME	SCORE
100	ORACLE	01
200	ORACLE	01

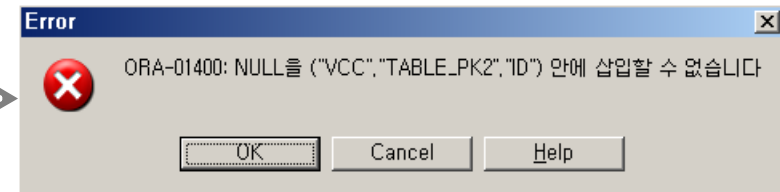
```
INSERT INTO TABLE_PK2
VALUES ('100','ORACLE','02');
```

'ID, SAME' 컬럼 조합 결과가 중복



```
INSERT INTO TABLE_PK2
VALUES (NULL,'ORACLE','01');
```

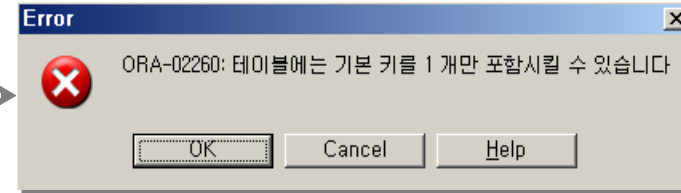
조합되는 개별 컬럼에 NULL은 허용되지 않음



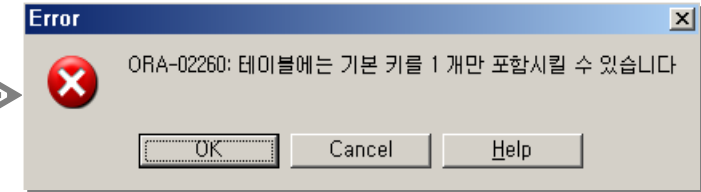
## 5.1.2 제약조건 - PRIMARY KEY : 조합 컬럼 생성 예

```
CREATE TABLE TABLE_PK3  
(ID      CHAR(3) PRIMARY KEY,  
 SNAME  VARCHAR2(20) PRIMARY KEY,  
 SCORE  CHAR(2));
```

'PRIMARY KEY' 키워드는 한번만 사용 가능



```
CREATE TABLE TABLE_PK3  
(ID      CHAR(3) CONSTRAINT PK1 PRIMARY KEY,  
 SNAME  VARCHAR2(20) CONSTRAINT PK1 PRIMARY KEY,  
 SCORE  CHAR(2));
```



동일한 제약조건 이름을 지정 → 컬럼 조합 결과를 대상으로 하는 제약조건 생성 의미가 아님

## 5.1.2 제약조건 - FOREIGN KEY

참조 테이블의 컬럼 값과 일치하거나 NULL 상태를 유지하도록 한다

EMPLOYEE	EMP_ID	EMP_NAME	DEPT_ID
	100	한선기	90
	101	강중훈	90
	102	최만식	90
	103	정도연	60
	104	안석규	60
	107	조재형	60
	124	정지현	50
	141	김예수	50
	143	나승원	50
	144	김순이	50
	149	성해교	50
	174	전우성	80
	176	엄정하	80
	178	심하균	
	200	고승우	10
	201	박하일	50
	202	권상후	10
	205	임영애	10
	206	엄정하	
	207	김술오	20
	208	이중기	20
	210	감우섭	20

DEPT_ID	DEPT_NAME	DEPARTMENT
20	회계팀	
10	본사 인사팀	
50	해외영업1팀	
60	기술지원팀	
80	해외영업2팀	
90	해외영업3팀	
30	마케팅팀	

- DEPT\_ID 컬럼 → FOREIGN KEY 컬럼
- DEPARTMENT 테이블의 DEPT\_ID 컬럼에 존재하지 않는 값이 포함되면 데이터 무결성에 문제가 있음

## 5.1.2 제약조건 - FOREIGN KEY : 컬럼 레벨에서 생성

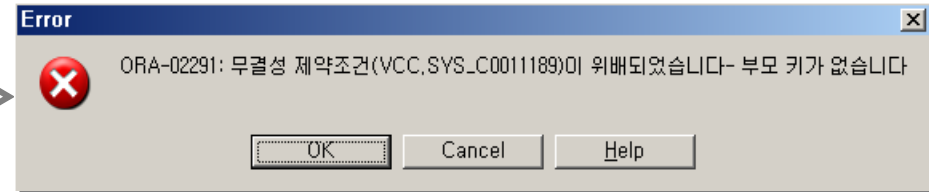
```
CREATE TABLE TABLE_FK
(ID      CHAR(3),
SNAME   VARCHAR2(20),
LID      CHAR(2) REFERENCES LOCATION ( LOCATION_ID ) );
```

참조 테이블만 기술하고 참조  
컬럼을 생략하면 해당 테이블의  
PRIMARY KEY 컬럼을 참조하게 됨

참조 테이블

참조 컬럼

```
INSERT INTO TABLE_FK
VALUES ( '200', 'ORACLE', 'C1' );
```



LOCATION_ID	COUNTRY_ID	LOC_DESCRIBE
A1	KO	아시아지역1
A2	JP	아시아지역2
A3	CH	아시아지역3
U1	US	미주지역
OT	ID	기타지역

LOCATION 테이블

참조 테이블 LOCATION에는  
LOCATION\_ID = 'C1'인 값이  
없음

## 5.1.2 제약조건 - FOREIGN KEY : 테이블 레벨에서 생성

테이블 레벨에서 생성하는 구문은 "FOREIGN KEY" 키워드가 추가됨

```
CREATE TABLE TABLE_FK2  
(ID CHAR(3),  
SNAME VARCHAR2(20),  
LID CHAR(2),  
[CONSTRAINT FK1] FOREIGN KEY ( LID ) REFERENCES LOCATION ( LOCATION_ID ) );
```

추가 키워드

설정 컬럼

참조 테이블

참조 컬럼

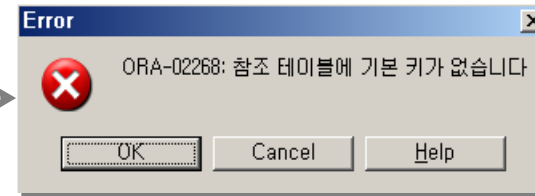
## 5.1.2 제약조건 - FOREIGN KEY : 생성 예

참조 테이블의 PRIMARY KEY/UNIQUE 제약조건이 설정된 컬럼만 참조 가능

```
CREATE TABLE TABLE_NOPK  
(ID CHAR(3),  
 SNAME VARCHAR2(20));
```

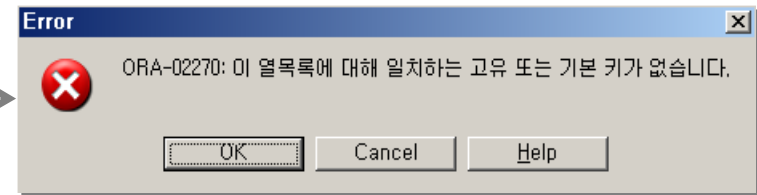
TABLE\_NOPK 테이블에는  
PRIMARY KEY 나 UNIQUE  
제약조건이 없음

```
CREATE TABLE TABLE_FK3  
(ID CHAR(3) REFERENCES TABLE_NOPK,  
 SNAME VARCHAR2(20));
```



참조 컬럼 이름을 생략했으므로 해당 테이블의 PRIMARY KEY 컬럼을 찾겠다는 의미  
→ PRIMARY KEY가 존재하지 않으므로 생성 불가

```
CREATE TABLE TABLE_FK3  
(ID CHAR(3) REFERENCES TABLE_NOPK(ID),  
 SNAME VARCHAR2(20));
```



참조 컬럼 ID에는 PRIMARY KEY나 UNIQUE 제약조건이 없음

### 5.1.2 제약조건 - FOREIGN KEY : DELETION OPTION

FOREIGN KEY 제약조건을 생성할 때, 참조 컬럼 값이 삭제되는 경우 FOREIGN KEY 컬럼 값을 어떻게 처리할 지 지정 가능

[구문]

```
[ CONSTRAINT constraint_name ] constraint_type ON DELETE SET NULL
```

또는

```
[ CONSTRAINT constraint_name ] constraint_type ON DELETE CASCADE
```

[구문 설명]

- **ON DELETE SET NULL**

참조 컬럼 값이 삭제될 때, FOREIGN KEY 컬럼 값을 NULL로 변경하는 OPTION

- **ON DELETE CASCADE**

참조 컬럼 값이 삭제될 때, FOREIGN KEY 컬럼 값도 함께 삭제(행 삭제 의미)하는 OPTION

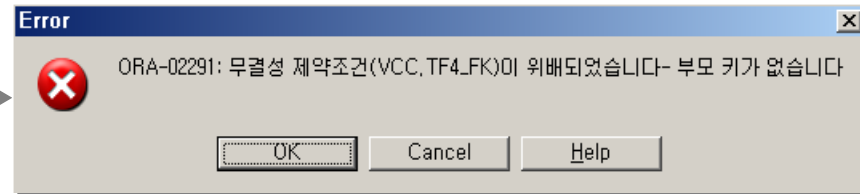


## 5.1.2 제약조건 - FOREIGN KEY : 조합 컬럼 생성 예

```
CREATE TABLE TABLE_FK4
(ID      CHAR(3),
SNAME   VARCHAR2(20),
SCODE   CHAR(2),
CONSTRAINT TF4_FK FOREIGN KEY ( ID, SNAME ) REFERENCES TABLE_PK2 );
```

조합 컬럼을 대상으로 FOREIGN KEY를 생성하려면 테이블 레벨에서 생성

```
INSERT INTO TABLE_FK4
VALUES ( '200', 'IBM', '03' );
```



ID	SNAME	SCODE
100	ORACLE	01
200	ORACLE	01

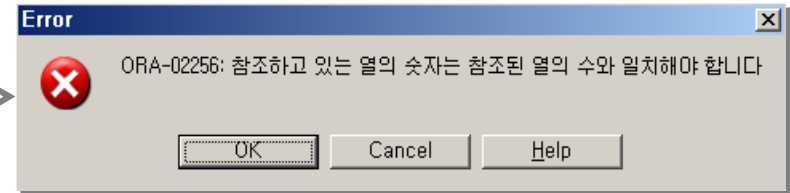
참조 테이블에 ('200','IBM') 조합 값이 없으므로 입력 불가

TABLE\_PK2 테이블  
PRIMARY KEY → (ID ,SNAME)

## 5.1.2 제약조건 - FOREIGN KEY : 생성 예

```
CREATE TABLE TABLE_FK5  
(ID      CHAR(3) REFERENCES TABLE_PK2,  
 SNAME  VARCHAR2(20) REFERENCES TABLE_PK2,  
 SCORE  CHAR(2));
```

TABLE\_PK2 테이블의 PRIMARY KEY는 (ID, SNAME) 컬럼 조합이므로 단일 컬럼은 FOREIGN KEY 제약조건을 생성할 수 없음



## 5.1.2 제약조건 - CHECK

각 컬럼 값이 만족해야 하는 조건을 지정

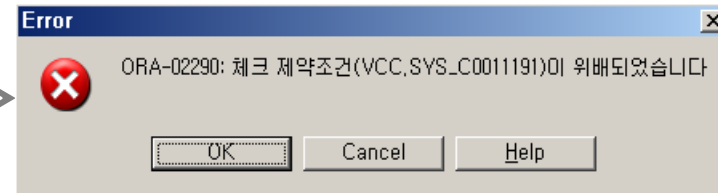
```
CREATE TABLE TABLE_CHECK  
(EMP_ID CHAR(3) PRIMARY KEY,  
  SALARY NUMBER CHECK ( SALARY > 0 ),  
  MARRIAGE CHAR(1),  
  CONSTRAINT CHK_MRG CHECK ( MARRIAGE IN ( 'Y','N' ) ) );
```

☞ SALARY 컬럼에는 0보다  
큰 값만 포함될 수 있음

☞ MARRIAGE 컬럼에는  
'Y'/'N'만 포함될 수 있음

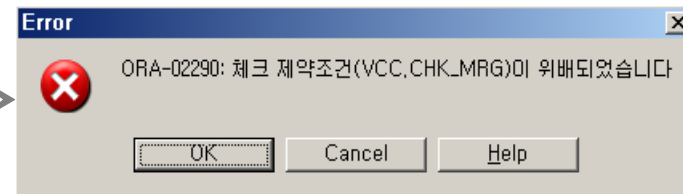
```
INSERT INTO TABLE_CHECK  
VALUES ('100', -100, 'Y');
```

SALARY = -100 이므로 CHECK  
제약조건에 위배



```
INSERT INTO TABLE_CHECK  
VALUES ('100', 500, '?');
```

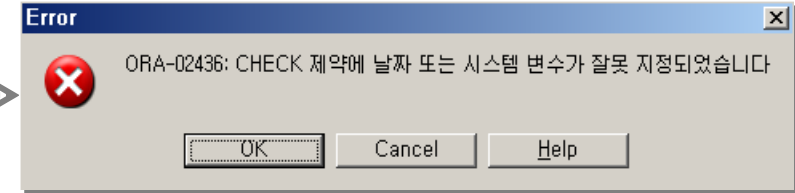
MARRIAGE = '?' 이므로 CHECK  
제약조건에 위배



## 5.1.2 제약조건 - CHECK : 사용 예

```
CREATE TABLE TABLE_CHECK2  
(ID          CHAR(3) PRIMARY KEY,  
 HIREDATE DATE CHECK ( HIREDATE < SYSDATE ) );
```

변하는 값은 조건으로 사용할 수 없음



```
CREATE TABLE TABLE_CHECK3  
(EID          CHAR(3) PRIMARY KEY,  
 ENAME        VARCHAR2(10) NOT NULL,  
 SALARY        NUMBER ,  
 MARRIAGE     CHAR(1),  
 CHECK ( SALARY > 0 AND SALARY < 1000000 ) );
```

CHECK 조건을 여러 개 사용할 수 있음

## 5.1.2 제약조건 - SAMPLE SCRIPT

```
CREATE TABLE CONSTRAINT_EMP
(EID          CHAR(3) CONSTRAINT PKEID PRIMARY KEY,
ENAME        VARCHAR2(20) CONSTRAINT NENAME NOT NULL,
ENO          CHAR(14) CONSTRAINT NENO NOT NULL CONSTRAINT UENO UNIQUE,
EMAIL        VARCHAR2(25) CONSTRAINT UEMAIL UNIQUE,
PHONE        VARCHAR2(12),
HIRE_DATE    DATE DEFAULT SYSDATE,
JID          CHAR(2) CONSTRAINT FKJID REFERENCES JOB ON DELETE SET NULL,
SALARY       NUMBER,
BONUS_PCT    NUMBER,
MARRIAGE     CHAR(1) DEFAULT 'N' CONSTRAINT CHK CHECK (MARRIAGE IN ('Y','N')),
MID          CHAR(3) CONSTRAINT FK MID REFERENCES CONSTRAINT_EMP ON DELETE SET NULL,
DID          CHAR(2),
CONSTRAINT FKDID FOREIGN KEY (DID) REFERENCES DEPARTMENT ON DELETE CASCADE
);
```

### 5.1.3 서브쿼리를 활용한 테이블 생성 구문

"AS SUBQUERY" 옵션을 사용하면 테이블 생성과 행 삽입을 동시에 할 수 있음

[구문]

```
CREATE TABLE table_name [ ( column_name [DEFAULT expr] [, ... ] ) ]  
AS SUBQUERY ;
```

[구문 설명]

- **특징**

테이블을 생성하고, 서브쿼리 실행 결과가 자동으로 입력됨

- **컬럼 정의**

- 데이터 타입 정의 불가 : 컬럼 이름 및 DEFAULT 값만 정의 가능
- 컬럼 이름 생략 가능 : 서브쿼리에서 사용한 컬럼 이름이 적용
- 제약조건 : 서브쿼리에서 사용한 대상 컬럼들의 NOT NULL 조건은 자동 반영됨
- 생성 시점에 컬럼 레벨에서 제약조건 생성 가능 → REFERENCES 제약조건 불가

## 5.1.3 서브쿼리를 활용한 테이블 생성 구문

```
CREATE TABLE TABLE_SUBQUERY1
AS SELECT EMP_ID, EMP_NAME, SALARY, DEPT_NAME, JOB_TITLE
FROM EMPLOYEE
LEFT JOIN DEPARTMENT USING (DEPT_ID)
LEFT JOIN JOB USING (JOB_ID);
```

[생성 결과]

Columns of TABLE_SUBQUERY1		
Name	Type	Nullable
▶ EMP_ID	CHAR(3)	Y
EMP_NAME	VARCHAR2(20)	
SALARY	NUMBER	Y
DEPT_NAME	VARCHAR2(30)	Y
JOB_TITLE	VARCHAR2(35)	Y

- EMP\_ID 컬럼은 원래 PRIMARY KEY 제약조건이 생성되어 있었으나 해당 제약조건은 자동으로 적용되지 않았으므로 NULL이 허용되는 상태임
- EMP\_NAME 컬럼은 설정된 NOT NULL 제약조건이 자동으로 적용된 상태임
- 컬럼 이름을 별도로 지정하지 않았으므로 서브쿼리에서 사용한 컬럼 이름이 적용됨

## 5.1.3 서브쿼리를 활용한 테이블 생성 구문

```
CREATE TABLE TABLE_SUBQUERY2 ( EID, ENAME, SALARY, DNAME, JTITLE )  
AS SELECT EMP_ID, EMP_NAME, SALARY, DEPT_NAME, JOB_TITLE  
FROM    EMPLOYEE  
LEFT JOIN    DEPARTMENT USING (DEPT_ID)  
LEFT JOIN    JOB USING (JOB_ID);
```

[생성 결과]

Columns of TABLE_SUBQUERY2		
Name	Type	Nullable
EID	CHAR(3)	Y
ENAME	VARCHAR2(20)	
SALARY	NUMBER	Y
DNAME	VARCHAR2(30)	Y
JTITLE	VARCHAR2(35)	Y

지정한 컬럼 이름으로 생성됨



## 5.1.3 서브쿼리를 활용한 테이블 생성 구문

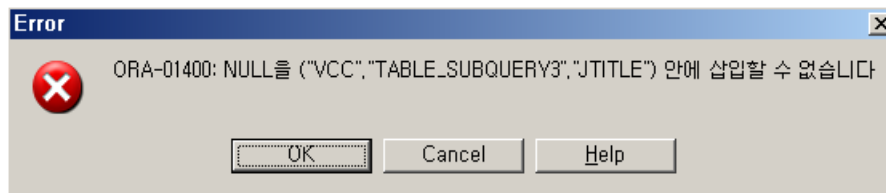
```
CREATE TABLE TABLE_SUBQUERY3  
( EID PRIMARY KEY,  
  ENAME,  
  SALARY CHECK (SALARY > 2000000),  
  DNAME,  
  JTITLE NOT NULL)  
AS SELECT EMP_ID, EMP_NAME, SALARY, DEPT_NAME, JOB_TITLE  
FROM EMPLOYEE  
LEFT JOIN DEPARTMENT USING (DEPT_ID)  
LEFT JOIN JOB USING (JOB_ID);
```



- SALARY > 2000000 조건 때문에 서브쿼리 실행 결과 중 2000000보다 작은 SALARY 값을 입력할 수 없음
- JTITLE 컬럼의 NOT NULL 조건 때문에 서브쿼리 실행 결과 중 JOB\_TITLE이 NULL인 값을 입력할 수 없음

## 5.1.3 서브쿼리를 활용한 테이블 생성 구문

```
CREATE TABLE TABLE_SUBQUERY3  
( EID PRIMARY KEY,  
  ENAME,  
  SALARY CHECK (SALARY > 2000000),  
  DNAME,  
  JTITLE NOT NULL)  
AS SELECT EMP_ID, EMP_NAME, SALARY, DEPT_NAME, JOB_TITLE  
FROM EMPLOYEE  
LEFT JOIN DEPARTMENT USING (DEPT_ID)  
LEFT JOIN JOB USING (JOB_ID)  
WHERE SALARY > 2000000;
```



- WHERE 조 건 에 SALARY > 2000000 조건을 추가하여 해결
- JTITLE 컬럼의 NOT NULL 조건 때문에 서브쿼리 실행 결과 중 JOB\_TITLE이 NULL인 값을 입력할 수 없음

## 5.1.3 서브쿼리를 활용한 테이블 생성 구문

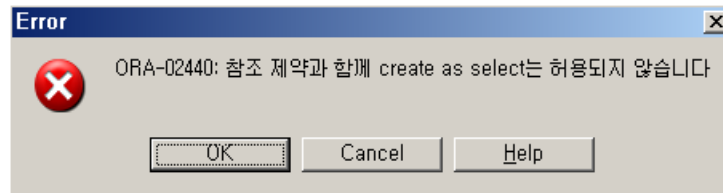
```
CREATE TABLE TABLE_SUBQUERY3
( EID PRIMARY KEY,
  ENAME,
  SALARY CHECK (SALARY > 20000000),
  DNAME,
  JTITLE DEFAULT 'N/A' NOT NULL)
AS SELECT EMP_ID, EMP_NAME, SALARY, DEPT_NAME, JOB_TITLE
FROM   EMPLOYEE
LEFT JOIN   DEPARTMENT USING (DEPT_ID)
LEFT JOIN   JOB USING (JOB_ID)
WHERE SALARY > 20000000;
```

Columns of TABLE_SUBQUERY3			
Name	Type	Nullable	Default
▶ EID	CHAR(3)		
ENAME	VARCHAR2(20)		
SALARY	NUMBER	Y	
DNAME	VARCHAR2(30)	Y	
JTITLE	VARCHAR2(35)		'N/A'

- EID 컬럼에 PRIMARY KEY 제약조건 적용됨
- JTITLE 컬럼의 DEFAULT 설정으로 해결됨

## 5.1.3 서브쿼리를 활용한 테이블 생성 구문

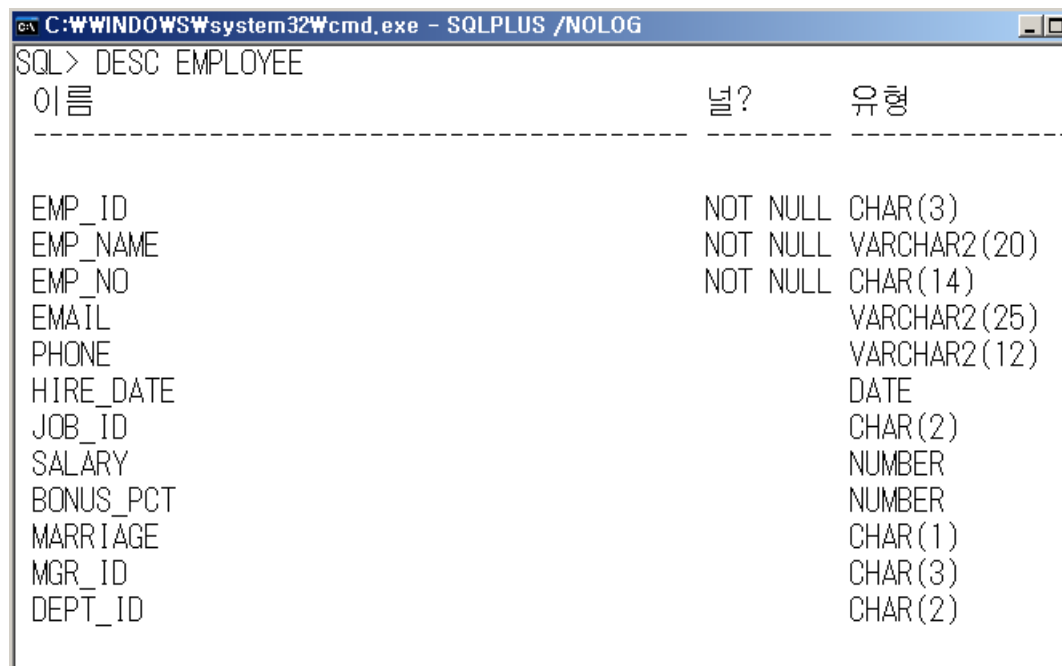
```
CREATE TABLE TABLE_SUBQUERY4  
( EID PRIMARY KEY,  
  ENAME ,  
  SALARY CHECK (SALARY > 20000000),  
  DID REFERENCES DEPARTMENT,  
  JTITLE DEFAULT 'N/A' NOT NULL)  
AS SELECT EMP_ID, EMP_NAME, SALARY, DEPT_ID, JOB_TITLE  
FROM    EMPLOYEE  
LEFT JOIN    JOB USING (JOB_ID)  
WHERE SALARY > 20000000;
```



REFERENCES 제약조건 사용 불가

## 5.1.4 테이블 구조 확인 - SQL\*Plus

```
SQL> DESC[RIBE] table_name
```



```
C:\WINDOWS\system32\cmd.exe - SQLPLUS /NOLOG
SQL> DESC EMPLOYEE
이름                                널?       유형
-----
EMP_ID                             NOT NULL   CHAR(3)
EMP_NAME                           NOT NULL   VARCHAR2(20)
EMP_NO                             NOT NULL   CHAR(14)
EMAIL                               NULL       VARCHAR2(25)
PHONE                               NULL       VARCHAR2(12)
HIRE_DATE                           NULL       DATE
JOB_ID                             NULL       CHAR(2)
SALARY                             NULL       NUMBER
BONUS_PCT                          NULL       NUMBER
MARRIAGE                           NULL       CHAR(1)
MGR_ID                             NULL       CHAR(3)
DEPT_ID                            NULL       CHAR(2)
```

컬럼 이름, NULL 포함 여부, 데이터 타입 확인 가능

## 5.1.4 테이블 구조 확인 - SQL\*Plus

활용 가능한 기타 SQL 구문

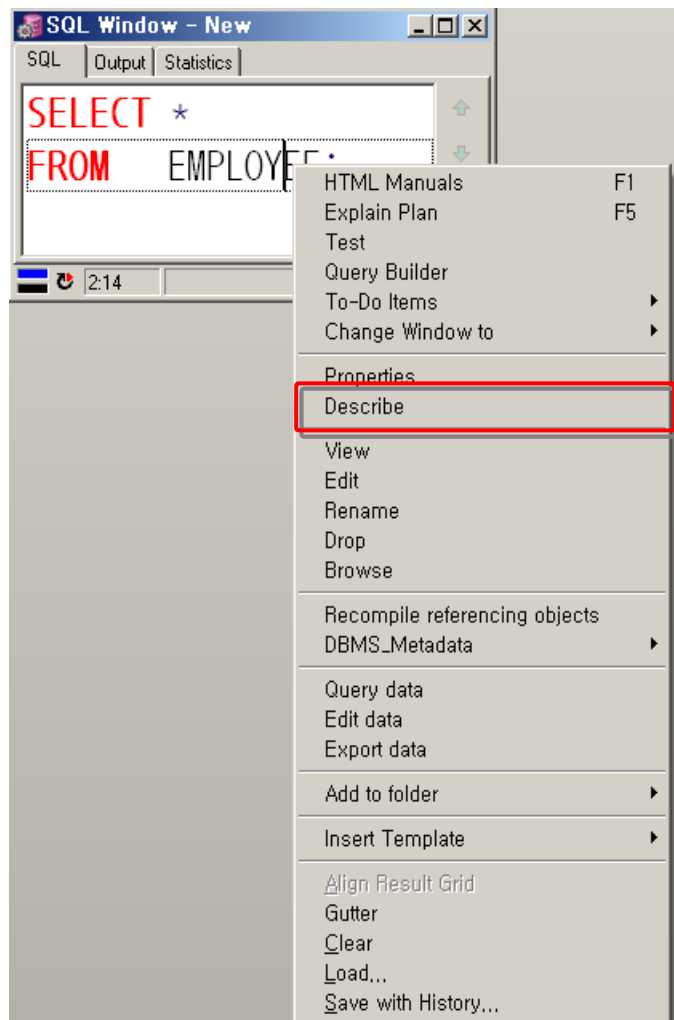
```
SQL> COL COLUMN_NAME FOR A15
SQL> COL DATA_TYPE FOR A15
SQL> COL DATA_DEFAULT A15
SQL> SELECT COLUMN_NAME,
2          DATA_TYPE,
3          DATA_DEFAULT,
4          NULLABLE
5 FROM USER_TAB_COLS
6 WHERE TABLE_NAME='EMPLOYEE';
```

COLUMN_NAME	DATA_TYPE	DATA_DEFAU	NULLABLE
EMP_ID	CHAR		N
EMP_NAME	VARCHAR2		N
EMP_NO	CHAR		N
EMAIL	VARCHAR2		Y
PHONE	VARCHAR2		Y
HIRE_DATE	DATE	SYSDATE	Y
JOB_ID	CHAR		Y
SALARY	NUMBER		Y
BONUS_PCT	NUMBER		Y
MARRIAGE	CHAR	'N'	Y
MGR_ID	CHAR		Y
DEPT_ID	CHAR		Y

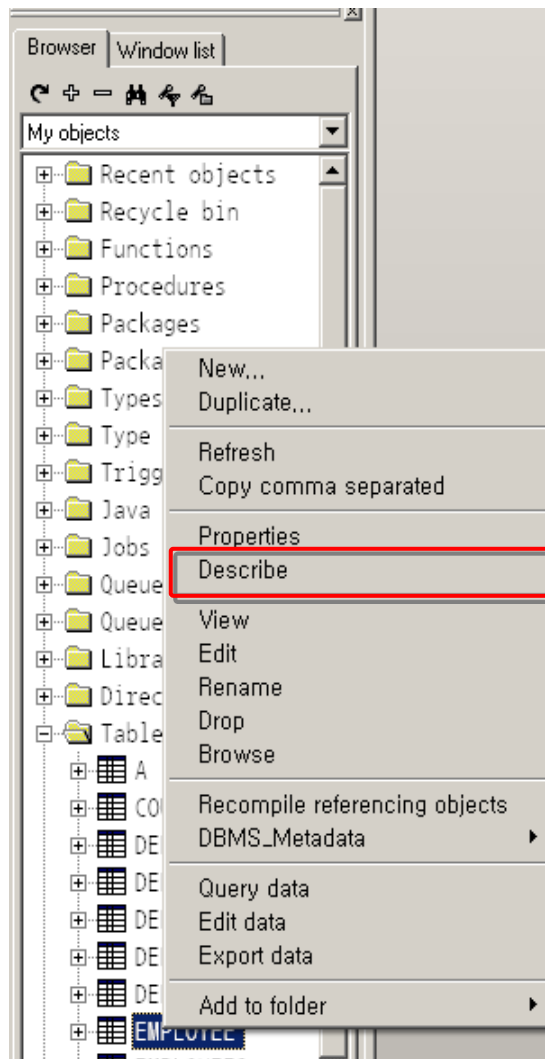
- COL 컬럼이름 FOR A15  
지정한 컬럼을 문자 15자리까지만 표시하는 명령
- 원하는 테이블 이름으로 조회 가능

## 5.1.4 테이블 구조 확인 - PL/SQL Developer

SQL 윈도우에서 코드 작성 시, 테이블 이름 부분에  
커서 위치 → 오른쪽 마우스 클릭 → DESCRIBE 선택



화면 왼쪽 Browser 창 → 테이블 이름 선택  
→ 오른쪽 마우스 클릭 → DESCRIBE 선택



## 5.1.4 테이블 구조 확인 - PL/SQL Developer

[결과]

Columns of EMPLOYEE				
Name	Type	Nullable	Default	Comments
▶EMP_ID	CHAR(3)			사번
EMP_NAME	VARCHAR2(20)			이름
EMP_NO	CHAR(14)			주민번호
EMAIL	VARCHAR2(25)	Y		e-Mail
PHONE	VARCHAR2(12)	Y		전화번호
HIRE_DATE	DATE	Y	SYSDATE	입사일
JOB_ID	CHAR(2)	Y		직급 코드
SALARY	NUMBER	Y		급여
BONUS_PCT	NUMBER	Y		보너스 지급율
MARRIAGE	CHAR(1)	Y	'N'	결혼 여부
MGR_ID	CHAR(3)	Y		관리자 사번
DEPT_ID	CHAR(2)	Y		부서 코드

컬럼 이름, NULL 포함 여부, 데이터 타입 이외에 DEFAULT 값, 컬럼 설명 까지 확인 가능



### 5.1.5 데이터 디렉터리 Data Dictionary

- 사용자 테이블
  - 일반 사용자가 생성하고 유지/관리하는 테이블
  - 사용자 데이터를 포함
- 데이터 디렉터리
  - 오라클 DBMS가 내부적으로 생성하고 유지/관리하는 테이블
  - 데이터베이스 정보(사용자 이름, 사용자 권한, 객체 정보, 제약 조건 등)를 포함
  - USER\_XXXXX 형식의 데이터 디렉터리에 접근 가능

## 5.1.5 데이터 디렉터리 - 테이블 이름 조회

```
SELECT TABLE_NAME  
FROM   USER_TABLES;
```

테이블 정보 관리

```
SELECT TABLE_NAME  
FROM   USER_CATALOG;
```

테이블, 뷰, 시퀀스 정보 관리

```
SELECT OBJECT_NAME  
FROM   USER_OBJECTS  
WHERE  OBJECT_TYPE = 'TABLE';
```

테이블, 뷰, 시퀀스, 인덱스 등 사용자 객체의 모든 정보 관리

결과(일부)

TABLE_NAME	
TABLE_NOTNULL	...
TABLE_UNIQUE	...
TABLE_UNIQUE2	...
TABLE_UNIQUE3	...
TABLE_UNIQUE5	...

결과(일부)

OBJECT_NAME	
TABLE_NOTNULL	...
TABLE_UNIQUE	...
TABLE_UNIQUE2	...
TABLE_UNIQUE3	...
TABLE_UNIQUE5	...

## 5.1.5 데이터 디렉터리 - 테이블 및 객체 정보 조회

```
SELECT OBJECT_TYPE AS 유형,
       COUNT(*) AS 개수
FROM   USER_OBJECTS
GROUP BY OBJECT_TYPE;
```

사용자 객체 별 개수 현황 확인

결과(일부)

유형	개수
PROCEDURE	8
TABLE	40
INDEX	23
FUNCTION	5

```
SELECT OBJECT_NAME AS 이름,
       OBJECT_TYPE AS 유형,
       CREATED AS 생성일,
       LAST_DDL_TIME AS 최종수정일
FROM   USER_OBJECTS
WHERE  OBJECT_TYPE = 'TABLE';
```

테이블 별 생성일, 수정일 현황 확인

결과(일부)

이름	유형	생성일	최종수정일
TABLE_NOTNULL	TABLE	09/12/11	09/12/11
TABLE_UNIQUE	TABLE	09/12/11	09/12/11
TABLE_UNIQUE2	TABLE	09/12/11	09/12/11
TABLE_UNIQUE3	TABLE	09/12/11	09/12/11
TABLE_UNIQUE5	TABLE	09/12/11	09/12/11

5.1.5 데이터 디렉터리 - 제약조건

USER\_CONSTRAINTS , <sup>주</sup>USER\_CONS\_COLUMNS를 이용

[USER\_CONSTRAINTS 구조(일부)]    [주요 항목 요약]

Name
OWNER
CONSTRAINT_NAME
CONSTRAINT_TYPE
TABLE_NAME
SEARCH_CONDITION
R_OWNER
R_CONSTRAINT_NAME
DELETE_RULE
STATUS

항목	설명
CONSTRAINT_TYPE	• P : PRIMARY KEY • U : UNIQUE • R : REFERENCES • C : CHECK, NOT NULL
SEARCH_CONDITION	CHECK 제약조건 내용
R_CONSTRAINT_NAME	참조 테이블의 PRIMARY KEY 이름
DELETE_RULE	참조 테이블의 PRIMARY KEY 컬럼이 삭제될 때 적용되는 규칙 "No Action", "SET NULL", "CASCADE" 로 표시

<sup>주</sup>USER\_CONS\_COLUMNS에서는 컬럼 이름을 확인할 수 있음

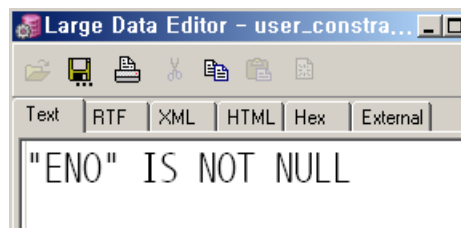
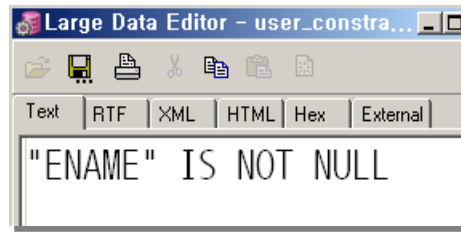
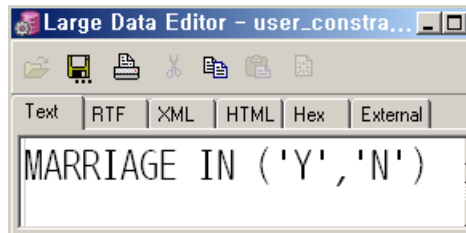
## 5.1.5 데이터 디렉터리 - 제약조건 확인

```

SELECT CONSTRAINT_NAME AS 이름,
       CONSTRAINT_TYPE AS 유형,
       R_CONSTRAINT_NAME AS 참조,
       DELETE_RULE AS 삭제규칙,
       SEARCH_CONDITION AS 내용
FROM   USER_CONSTRAINTS
WHERE  TABLE_NAME='CONSTRAINT_EMP';

```

이름	유형	참조	삭제규칙	내용
NENAME	C			<Long>
NENO	C			<Long>
CHK	C			<Long>
PKEID	P			<Long>
UENO	U			<Long>
UEMAIL	U			<Long>
FKJID	R	PK_JOBID	SET NULL	<Long>
FKMID	R	PKEID	SET NULL	<Long>
FKDID	R	PK_DEPTID	CASCADE	<Long>



'NOT NULL' 제약조건은  
CHECK 제약조건 유형으로  
관리됨

## 5.1.5 데이터 디렉터리 - 제약조건 확인

```

SELECT CONSTRAINT_NAME AS 이름,
       CONSTRAINT_TYPE AS 유형,
       COLUMN_NAME AS 컬럼,
       R_CONSTRAINT_NAME AS 참조,
       DELETE_RULE AS 삭제규칙,
       SEARCH_CONDITION AS 내용
FROM   USER_CONSTRAINTS
JOIN   USER_CONS_COLUMNS USING (CONSTRAINT_NAME, TABLE_NAME)
WHERE  TABLE_NAME='CONSTRAINT_EMP';

```

이름	유형	컬럼	참조	삭제규칙	내용
NENAME	C	ENAME			<Long>
NENO	C	ENO			<Long>
CHK	C	MARRIAGE			<Long>
PKEID	P	EID			<Long>
UENO	U	ENO			<Long>
UEMAIL	U	EMAIL			<Long>
FKJID	R	JID	PK_JOBID	SET NULL	<Long>
FKMID	R	MID	PKEID	SET NULL	<Long>
FKDID	R	DID	PK_DEPTID	CASCADE	<Long>

### 5.1.6 테이블 수정 - 범위

- 컬럼 관련

- 컬럼 추가/삭제
- 데이터 타입 변경, DEFAULT 변경
- 컬럼 이름 변경

- 제약조건 관련

- 제약조건 추가/삭제
- 제약조건 이름 변경

- 테이블 관련

- 테이블 이름 변경

## 5.1.6 테이블 수정 - 구문

[구문]

```
ALTER TABLE table_name
ADD ( column_name datatype [DEFAULT expr] [, ...] ) |
{ ADD [CONSTRAINT constraint_name] constraint_type (column_name) } ... |
MODIFY ( column_name datatype [DEFAULT expr] [, ...] ) |
DROP { [COLUMN column_name] | (column_name, ...) } [CASCADE CONSTRAINTS] |
DROP PRIMARY KEY [CASCADE] |
    { UNIQUE (column_name, ...) [CASCADE] } ... |
    CONSTRAINT constraint_name [CASCADE];
```

[이름 변경 구문]

```
ALTER TABLE old_table_name RENAME TO new_table_name ;
RENAME old_table_name TO new_table_name ;

ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name ;
ALTER TABLE table_name RENAME CONSTRAINT old_const_name TO new_const_name ;
```



## 5.1.6 테이블 수정 - 컬럼 추가

추가되는 컬럼은 테이블의 맨 마지막에 위치하며, 생성 위치를 변경할 수 없음

```
ALTER TABLE DEPARTMENT
ADD ( MGR_ID CHAR(3) );
```

Name	Type	Nullable
DEPT_ID	CHAR(2)	
DEPT_NAME	VARCHAR2(30)	Y
LOC_ID	CHAR(2)	
MGR_ID	CHAR(3)	Y

DEPT_ID	DEPT_NAME	LOC_ID	MGR_ID
20	회계팀	A1	
10	본사 인사팀	A1	
50	해외영업1팀	U1	
60	기술지원팀	OT	
80	해외영업2팀	A2	
90	해외영업3팀	A3	
30	마케팅팀	A1	

DEFAULT 값이 없으면 추가되는 컬럼은 NULL이 적용됨

```
ALTER TABLE DEPARTMENT
ADD ( MGR_ID CHAR(3) DEFAULT '101' );
```

Name	Type	Nullable	Default
DEPT_ID	CHAR(2)		
DEPT_NAME	VARCHAR2(30)	Y	
LOC_ID	CHAR(2)		
MGR_ID	CHAR(3)	Y	'101'

DEPT_ID	DEPT_NAME	LOC_ID	MGR_ID
20	회계팀	A1	101
10	본사 인사팀	A1	101
50	해외영업1팀	U1	101
60	기술지원팀	OT	101
80	해외영업2팀	A2	101
90	해외영업3팀	A3	101
30	마케팅팀	A1	101

DEFAULT 값을 설정하면 추가되는 컬럼에 DEFAULT 값이 적용됨

## 5.1.6 테이블 수정 - 제약조건 추가

- 'NOT NULL' 이외의 제약조건은 **ADD** 구문 사용(테이블 레벨에서의 정의 구문과 유사)
- 'NOT NULL' 제약조건은 **MODIFY** 구문 사용

```
CREATE TABLE EMP3 AS SELECT * FROM EMPLOYEE;
ALTER TABLE EMP3
ADD PRIMARY KEY (EMP_ID)
ADD UNIQUE (EMP_NO)
MODIFY HIRE_DATE NOT NULL;
```

☞ 샘플 테이블 EMP3 생성

☞ EMP\_ID 컬럼에 PRIMARY KEY 제약조건 추가

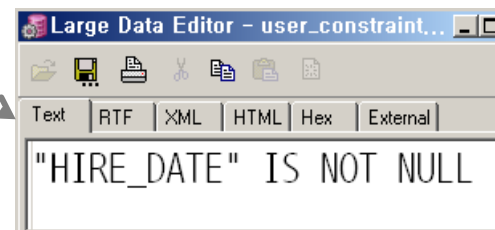
☞ EMP\_NO 컬럼에 UNIQUE 제약조건 추가

☞ HIRE\_DATE 컬럼에 NOT NULL 제약조건 추가

[제약조건 추가 결과 확인]

이름	유형	컬럼	참조	삭제규칙	내용
SYS_C0011246	C	EMP_NAME			<Long>
SYS_C0011247	C	EMP_NO			<Long>
SYS_C0011248	C	HIRE_DATE			<Long>
SYS_C0011249	P	EMP_ID			<Long>
SYS_C0011250	U	EMP_NO			<Long>

EMPLOYEE 테이블에서 상속된 제약조건



### 5.1.6 테이블 수정 - 컬럼 수정

- 컬럼 데이터 타입 관련

- 대상 컬럼이 비어있는 경우에만 타입 변경 가능
- 단, 컬럼 크기를 변경하지 않거나 증가시키는 경우 CHAR $\Leftrightarrow$ VARCHAR2 변환 가능

- 컬럼 크기 관련

- 대상 컬럼이 비어있는 경우 또는 테이블 전체에 데이터가 없는 경우 크기 감소 가능
- 포함된 데이터에 영향을 미치지 않는 범위에서는 크기 감소 가능

- DEFAULT 관련

DEFAULT 값이 변경되면 변경 이후부터 적용

## 5.1.6 테이블 수정 - 컬럼 수정 예

```
CREATE TABLE EMP4
AS SELECT EMP_ID, EMP_NAME, HIRE_DATE
FROM EMPLOYEE;
```

Name	Type
EMP_ID	CHAR(3)
EMP_NAME	VARCHAR2(20)
HIRE_DATE	DATE

샘플 테이블 생성

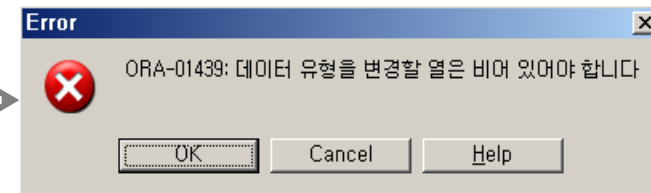
```
ALTER TABLE EMP4
MODIFY (EMP_ID VARCHAR2(5),
        EMP_NAME CHAR(20));
```

Name	Type
EMP_ID	VARCHAR2(5)
EMP_NAME	CHAR(20)
HIRE_DATE	DATE

EMP\_ID 컬럼: 크기를 증가시켰으므로 VARCHAR2 타입 변환 가능

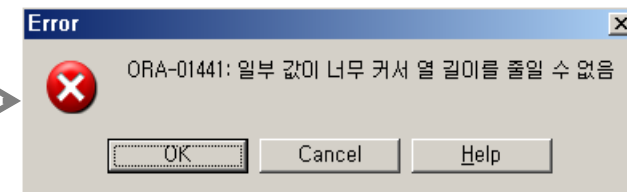
EMP\_NAME 컬럼: 크기 변경 없으므로 CHAR 타입 변환 가능

```
ALTER TABLE EMP4
MODIFY (HIRE_DATE CHAR(8));
```



DATE 타입을 CHAR 타입으로 변환하려면 데이터가 없어야 함

```
ALTER TABLE EMP4
MODIFY (EMP_NAME CHAR(15));
```



CHAR(20)에서 크기를 감소시키는 것은 불가

## 5.1.6 테이블 수정 - 컬럼 수정 예

```

CREATE TABLE EMP5
(EMP_ID    CHAR(3),
 EMP_NAME  VARCHAR2(20),
 ADDR1     VARCHAR2(20) DEFAULT '서울',
 ADDR2     VARCHAR2(100));

INSERT INTO EMP5
VALUES ('A10', '임태희', DEFAULT, '청담동');

INSERT INTO EMP5
VALUES ('B10', '이병언', DEFAULT, '분당 정자동');

SELECT * FROM EMP5;

```

```

ALTER TABLE EMP5
MODIFY (ADDR1 DEFAULT '경기');

INSERT INTO EMP5
VALUES ('C10', '임승우', DEFAULT, '분당 효자촌');

SELECT * FROM EMP5;

```

DEFAULT 값은 변경 이후부터 적용

Name	Type	Nullable	Default
EMP_ID	CHAR(3)	Y	
EMP_NAME	VARCHAR2(20)	Y	
ADDR1	VARCHAR2(20)	Y	'서울'
ADDR2	VARCHAR2(100)	Y	

EMP_ID	EMP_NAME	ADDR1	ADDR2
A10	임태희	서울	청담동
B10	이병언	서울	분당 정자동

Name	Type	Nullable	Default
EMP_ID	CHAR(3)	Y	
EMP_NAME	VARCHAR2(20)	Y	
ADDR1	VARCHAR2(20)	Y	'경기'
ADDR2	VARCHAR2(100)	Y	

EMP_ID	EMP_NAME	ADDR1	ADDR2
A10	임태희	서울	청담동
B10	이병언	서울	분당 정자동
C10	임승우	경기	분당 효자촌

## 5.1.6 테이블 수정 - 컬럼 삭제

- 컬럼 하나를 삭제 가능
- 컬럼 여러 개를 한번에 삭제 가능(구문이 달라짐)
- 주의 사항
  - 삭제 대상 컬럼에 데이터가 포함되어 있어도 삭제됨
  - 삭제 작업 후에는 테이블에 반드시 컬럼이 하나 이상 남아 있어야 함  
→ 모든 컬럼을 삭제할 수 없음
  - 삭제된 컬럼은 복구할 수 없음

```
ALTER TABLE EMP4  
DROP COLUMN EMP_ID;
```

또는

```
ALTER TABLE EMP4  
DROP (EMP_ID);
```

단일 컬럼의 삭제 구문은 'COLUMN'  
키워드나 ( ) 사용 가능

```
ALTER TABLE EMP5  
DROP (EMP_ID, EMP_NAME);
```

여러 컬럼의 삭제 구문은 ( ) 사용

## 5.1.6 테이블 수정 - 컬럼 삭제

## "CASCADE CONSTRAINTS" OPTION

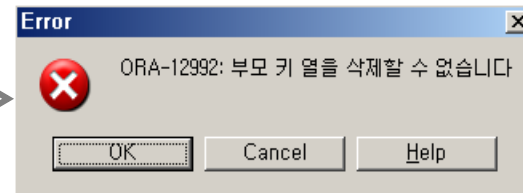
삭제되는 컬럼을 참조하고 있는 다른 컬럼에 설정된 제약조건까지 함께 삭제

```
CREATE TABLE TB1  
(PK    NUMBER PRIMARY KEY,  
  FK    NUMBER REFERENCES TB1,  
  COL1  NUMBER,  
  CHECK ( PK > 0 AND COL1 > 0 ));
```

```
ALTER TABLE TB1  
DROP (PK) CASCADE CONSTRAINTS;  
  
ALTER TABLE TB1  
DROP (COL1) CASCADE CONSTRAINTS;
```

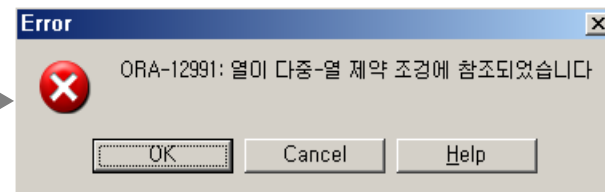
```
ALTER TABLE TB1 DROP (PK);
```

FK 컬럼에 PK 컬럼을 참조하는 REFERENCES  
제약조건이 설정되어 있으므로 삭제 불가



```
ALTER TABLE TB1 DROP (COL1);
```

COL1 컬럼에 CHECK 제약조건이 설정되어  
있으므로 삭제 불가



## 5.1.6 테이블 수정 - 제약조건 삭제 예

[CONSTRAINT\_EMP 테이블 제약조건 현황]

이름	유형	컬럼	참조	삭제규칙	내용
NENAME	C	ENAME			<Long>
NENO	C	ENO			<Long>
CHK	C	MARRIAGE			<Long>
PKEID	P	EID			<Long>
UENO	U	ENO			<Long>
UEMAIL	U	EMAIL			<Long>
FKJID	R	JID	PK_JOBID	SET NULL	<Long>
FKMID	R	MID	PKEID	SET NULL	<Long>
FKDID	R	DID	PK_DEPTID	CASCADE	<Long>

[CHECK, REFERENCES 유형 제약조건 삭제 예]

- CHECK 제약조건

```
ALTER TABLE CONSTRAINT_EMP
DROP CONSTRAINT CHK;
```

- REFERENCE 제약조건

```
ALTER TABLE CONSTRAINT_EMP
DROP CONSTRAINT FKJID
DROP CONSTRAINT FKMID
DROP CONSTRAINT FKDID;
```

CHECK, REFERENCES 유형 제약조건 삭제

- "DROP CONSTRAINT" 키워드와 제약조건 이름을 기술하여 삭제 가능



## 5.1.6 테이블 수정 - 제약조건 삭제 예

[CONSTRAINT\_EMP 테이블 제약조건 현황]

이름	유형	컬럼	참조	삭제규칙	내용
NENAME	C	ENAME			<Long>
NENO	C	ENO			<Long>
CHK	C	MARRIAGE			<Long>
PKEID	P	EID			<Long>
UENO	U	ENO			<Long>
UEMAIL	U	EMAIL			<Long>
FKJID	R	JID	PK_JOBID	SET NULL	<Long>
FKMID	R	MID	PKEID	SET NULL	<Long>
FKDID	R	DID	PK_DEPTID	CASCADE	<Long>

[PRIMARY KEY 유형 제약조건 삭제 예]

```
ALTER TABLE CONSTRAINT_EMP
DROP CONSTRAINT PK_EMPID [CASCADE];
```

또는

```
ALTER TABLE CONSTRAINT_EMP
DROP PRIMARY KEY [CASCADE];
```

PRIMARY KEY 제약조건 삭제

- "DROP CONSTRAINT" 키워드와 제약조건 이름을 기술하여 삭제 가능
- "DROP PRIMARY KEY" 구문으로도 삭제 가능

## 5.1.6 테이블 수정 - 제약조건 삭제 예

[CONSTRAINT\_EMP 테이블 제약조건 현황]

이름	유형	컬럼	참조	삭제규칙	내용
NENAME	C	ENAME			<Long>
NENO	C	ENO			<Long>
CHK	C	MARRIAGE			<Long>
PKEID	P	EID			<Long>
UENO	U	ENO			<Long>
UEMAIL	U	EMAIL			<Long>
FKJID	R	JID	PK_JOBID	SET NULL	<Long>
FKMID	R	MID	PKEID	SET NULL	<Long>
FKDID	R	DID	PK_DEPTID	CASCADE	<Long>

[UNIQUE 유형 제약조건 삭제 예]

```
ALTER TABLE CONSTRAINT_EMP
DROP CONSTRAINT UENO [CASCADE]
DROP CONSTRAINT UEMAIL [CASCADE];
```

또는

```
ALTER TABLE CONSTRAINT_EMP
DROP UNIQUE (ENO) [CASCADE]
DROP UNIQUE (EMAIL) [CASCADE];
```

UNIQUE 제약조건 삭제

- "DROP CONSTRAINT" 키워드와 제약조건 이름을 기술하여 삭제 가능
- "DROP UNIQUE (컬럼이름)" 구문으로도 삭제 가능

## 5.1.6 테이블 수정 - 제약조건 삭제 예

[CONSTRAINT\_EMP 테이블 제약조건 현황]

이름	유형	컬럼	참조	삭제규칙	내용
NENAME	C	ENAME			<Long>
NENO	C	ENO			<Long>
CHK	C	MARRIAGE			<Long>
PKEID	P	EID			<Long>
UENO	U	ENO			<Long>
UEMAIL	U	EMAIL			<Long>
FKJID	R	JID	PK_JOBID	SET NULL	<Long>
FKMID	R	MID	PKEID	SET NULL	<Long>
FKDID	R	DID	PK_DEPTID	CASCADE	<Long>

[NOT NULL 제약조건 삭제 예]

```
ALTER TABLE CONSTRAINT_EMP
DROP CONSTRAINT NENAME
DROP CONSTRAINT NENO;

또는

ALTER TABLE CONSTRAINT_EMP
MODIFY (ENAME NULL, ENO NULL);
```

NOT NULL 제약조건 삭제


- "DROP CONSTRAINT" 키워드와 제약조건 이름을 기술하여 삭제 가능
- "MODIFY (컬럼이름 NULL)" 구문으로도 삭제 가능

## 5.1.6 테이블 수정 - 이름 변경

```
CREATE TABLE TB_EXAM
(COL1 CHAR(3) PRIMARY KEY,
 ENAME VARCHAR2(20)
 FOREIGN KEY (COL1) REFERENCES EMPLOYEE);
```

샘플 테이블 생성


```
SELECT COLUMN_NAME
FROM   USER_TAB_COLS
WHERE  TABLE_NAME = 'TB_EXAM';
```



COLUMN_NAME
COL1
ENAME

컬럼 이름 조회

```
SELECT CONSTRAINT_NAME AS 이름,
       CONSTRAINT_TYPE AS 유형,
       COLUMN_NAME AS 컬럼,
       R_CONSTRAINT_NAME AS 참조,
       DELETE_RULE AS 삭제규칙
FROM   USER_CONSTRAINTS
JOIN   USER_CONS_COLUMNS
USING (CONSTRAINT_NAME, TABLE_NAME)
WHERE  TABLE_NAME='TB_EXAM';
```



이름	유형	컬럼	참조	삭제규칙
SYS_C0011409	P	COL1		
SYS_C0011410	R	COL1	PK_EMPID	NO ACTION

제약조건 현황 조회

## 5.1.6 테이블 수정 - 이름 변경

```
ALTER TABLE TB_EXAM
RENAME COLUMN COL1 TO EMPID;
```

컬럼 이름 변경

COLUMN_NAME
EMPID
ENAME

```
ALTER TABLE TB_EXAM
RENAME CONSTRAINTS SYS_C0011409 TO PK_EID;
```

제약조건 이름 변경

이름	유형	컬럼	참조	삭제규칙
PK_EID	P	EMPID		
SYS_C0011410	R	EMPID	PK_EMPID	NO ACTION

```
ALTER TABLE TB_EXAM
RENAME CONSTRAINTS SYS_C0011410 TO FK_EID;
```

제약조건 이름 변경

이름	유형	컬럼	참조	삭제규칙
PK_EID	P	EMPID		
FK_EID	R	EMPID	PK_EMPID	NO ACTION

```
ALTER TABLE TB_EXAM RENAME TO TB_SAMPLE;

또는

RENAME TB_EXAM TO TB_SAMPLE;
```

테이블 이름 변경

## 5.1.6 테이블 삭제

```
DROP TABLE table_name [CASCADE CONSTRAINTS];
```

[구문 설명]

- 포함된 데이터 및 테이블과 관련된 데이터 디렉터리 정보까지 모두 삭제
- 삭제 작업은 복구할 수 없음
- CASCADE CONSTRAINTS
  - 삭제 대상 테이블의 PRIMARY KEY 또는 UNIQUE 제약조건을 참조하는 다른 제약조건을 삭제하는 OPTION
  - 참조중인 제약조건이 있는 경우 OPTION 미 사용시 삭제할 수 없음

## 5.1.6 테이블 삭제

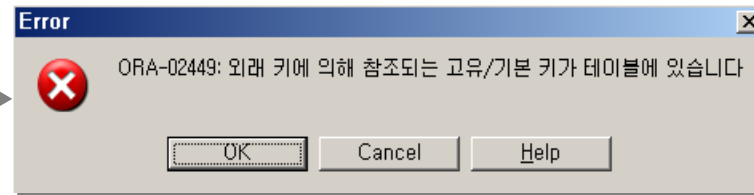
```
CREATE TABLE DEPT  
(DID CHAR(2) PRIMARY KEY,  
  DNAME VARCHAR2(10));  
  
CREATE TABLE EMP6  
(EID CHAR(3) PRIMARY KEY,  
  ENAME VARCHAR2(10),  
  DID CHAR(2) REFERENCES DEPT);
```

샘플 테이블 생성

```
DROP TABLE DEPT CASCADE CONSTRAINTS;
```

```
DROP TABLE DEPT;
```

EMP6 테이블의 DID 컬럼이  
DEPT 테이블의 DID 컬럼을  
참조하고 있으므로 삭제  
불가



# Ch 5.

## DDL(Data Definition Language)

5.1 | 테이블 Table

5.2 | 뷰 View

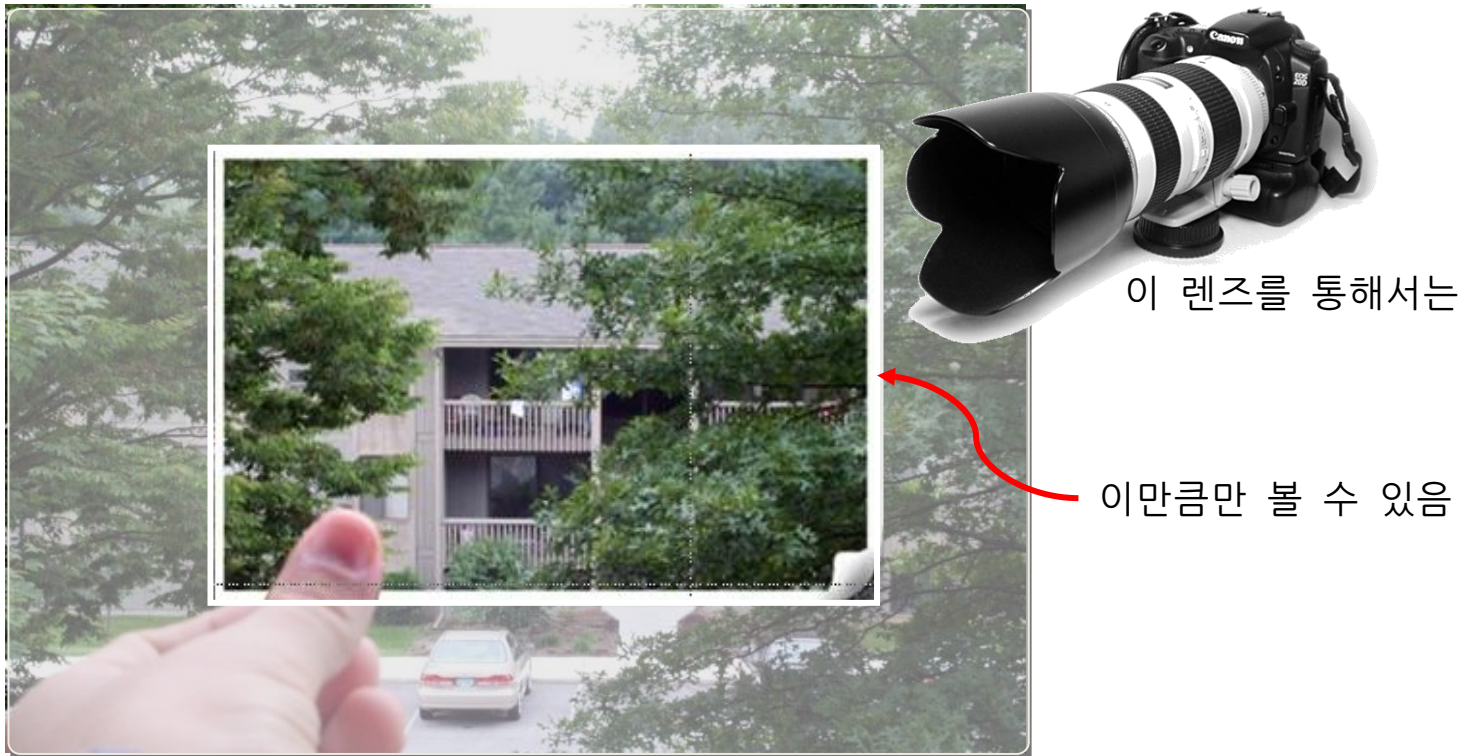
5.3 | 시퀀스 Sequence

5.4 | 인덱스 Index



### 5.2.1 뷰 - 개요

- 다른 테이블이나 뷰에 포함된 데이터의 맞춤표현 Tailored Presentation
- "STORED QUERY" 또는 "VIRTUAL TABLE" 로 간주되는 데이터베이스 객체



## 5.2.1 뷰 - 개념

- 하나 또는 하나 이상의 테이블/뷰에 포함된 데이터 부분 집합을 나타내는 논리적인 객체 → 선택적인 정보만 제공 가능
- 자체적으로 데이터를 포함하지 않는다
- 베이스 테이블<sup>Base Table</sup>: 뷰를 통해 보여지는 데이터를 포함하고 있는 실제 테이블

베이스 테이블 1(일부)

EMP_ID	EMP_NO	EMP_NAME	SALARY
100	621133-1483658	한선기	9000000
101	621136-1006405	강중훈	5500000
102	861011-1940062	최만식	3600000
103	631127-2519077	정도연	2600000
104	651031-1962810	안석규	3500000

뷰1(일부)

EMP_ID	EMP_NAME
100	한선기
101	강중훈
102	최만식
103	정도연
104	안석규

베이스 테이블 2(일부)

EMP_ID	EMP_NAME	DEPT_ID
100	한선기	...
101	강중훈	90
102	최만식	90
103	정도연	60
104	안석규	60

뷰 2(일부)

EMP_NAME	DEPT_NAME
강중훈	해외영업3팀
최만식	해외영업3팀
정도연	기술지원팀
안석규	기술지원팀
조재형	기술지원팀

베이스 테이블 3(일부)

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀

### 5.2.2 뷰 - 사용 목적 및 장점

- Restricted data access

뷰에 접근하는 사용자는 미리 정의된 결과만을 볼 수 있음 → 데이터 접근을 제한함으로써 중요한 데이터를 보호할 수 있음

- Hide data complexity

여러 테이블을 조인하는 등 복잡한 SQL 구문을 사용해야 하는 경우 자세한 SQL 구문의 내용을 숨길 수 있음

- Simplify statement for the user

복잡한 SQL 구문을 모르는 사용자라도 SELECT 구문만으로 원하는 결과를 조회할 수 있음

- Present the data in a different perspective

뷰에 포함되는 컬럼은 참조 대상 테이블에 영향을 주지 않고 다른 이름으로 참조 가능

- Isolate applications from changes in definitions of base tables

베이스 테이블에 포함된 여러 개 컬럼 중 일부만 사용하도록 뷰를 생성한 경우, 뷰가 참조하지 않는 나머지 컬럼이 변경되어도 뷰를 사용하는 다른 프로그램들은 영향을 받지 않음

- Save complex queries

복잡한 SQL 문을 뷰 형태로 저장하여 반복적으로 사용 가능

## 5.2.3 뷰 - 생성 구문

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name [( alias [, alias ...])]  
AS Subquery  
[WITH CHECK OPTION [ CONSTRAINT constraint_name ]]  
[WITH READ ONLY [ CONSTRAINT constraint_name ]] ;
```

[구문 설명]

- CREATE OR REPLACE  
지정한 이름의 뷰가 없으면 새로 생성, 동일 이름이 존재하면 수정<sup>Overwrite</sup>
- FORCE | NOFORCE
  - NOFORCE : 베이스 테이블이 존재하는 경우에만 뷰 생성 가능
  - FORCE : 베이스 테이블이 존재하지 않아도 뷰 생성 가능
- ALIAS
  - 뷰에서 사용할 표현식 이름(테이블 컬럼 이름 의미)
  - 생략 : SUBQUERY에서 사용한 이름 적용
  - ALIAS 개수 : SUBQUERY에서 사용한 SELECT LIST 개수와 일치
- SUBQUERY  
뷰에서 표현하는 데이터를 생성하는 SELECT 구문
- 제약 조건
  - WITH CHECK OPTION : 뷰를 통해 접근 가능한 데이터에 대해서만 DML 작업 허용
  - WITH READ ONLY : 뷰를 통해 DML 작업 허용 안 함
  - 제약조건으로 간주되므로 별도 이름 지정 가능

## 5.2.3 뷰 - 생성 예

- 뷰를 생성할 때 사용하는 서브쿼리는 일반적인 SELECT 구문을 사용
- 생성된 뷰는 테이블처럼 취급됨

```
CREATE OR REPLACE VIEW V_EMP
AS SELECT EMP_NAME, DEPT_ID
   FROM   EMPLOYEE
   WHERE  DEPT_ID = '90';
```

```
SELECT *
FROM   V_EMP;
```

EMP_NAME	DEPT_ID
한선기	90
강중훈	90
최만식	90

```
SELECT COLUMN_NAME, DATA_TYPE, NULLABLE
FROM   USER_TAB_COLS
WHERE  TABLE_NAME = 'V_EMP';
```

COLUMN_NAME	DATA_TYPE	NULLABLE
EMP_NAME	VARCHAR2	N
DEPT_ID	CHAR	Y

## 5.2.3 뷰 - 생성 예

```
CREATE OR REPLACE VIEW V_EMP_DEPT_JOB
AS SELECT EMP_NAME,
          DEPT_NAME,
          JOB_TITLE
FROM      EMPLOYEE
LEFT JOIN DEPARTMENT USING (DEPT_ID)
LEFT JOIN JOB USING (JOB_ID)
WHERE     JOB_TITLE = '사원';
```

```
SELECT *
FROM   V_EMP_DEPT_JOB;
```

EMP_NAME	DEPT_NAME	JOB_TITLE
정지현	해외영업1팀	사원
염정하		사원
성해교	해외영업1팀	사원
고승우	본사 인사팀	사원

```
SELECT COLUMN_NAME, DATA_TYPE, NULLABLE
FROM   USER_TAB_COLS
WHERE  TABLE_NAME = 'V_EMP_DEPT_JOB';
```

COLUMN_NAME	DATA_TYPE	NULLABLE
EMP_NAME	VARCHAR2	N
DEPT_NAME	VARCHAR2	Y
JOB_TITLE	VARCHAR2	Y

## 5.2.3 뷰 - 생성 예 : ALIAS 사용

- 뷰 정의 부분에서 지정 가능
- 서브쿼리 부분에서 지정 가능

```
CREATE OR REPLACE VIEW V_EMP_DEPT_JOB (ENM, DNM, TITLE)
AS SELECT EMP_NAME, DEPT_NAME, JOB_TITLE
   FROM   EMPLOYEE
   LEFT JOIN DEPARTMENT USING (DEPT_ID)
   LEFT JOIN JOB USING (JOB_ID)
   WHERE  JOB_TITLE = '사원';
```

```
CREATE OR REPLACE VIEW V_EMP_DEPT_JOB
AS SELECT EMP_NAME AS ENM,
          DEPT_NAME AS DNM,
          JOB_TITLE AS TITLE
   FROM EMPLOYEE
   LEFT JOIN DEPARTMENT USING (DEPT_ID)
   LEFT JOIN JOB USING (JOB_ID)
   WHERE  JOB_TITLE = '사원';
```

COLUMN_NAME	DATA_TYPE	NULLABLE
ENM	VARCHAR2	N
DNM	VARCHAR2	Y
TITLE	VARCHAR2	Y

## 5.2.3 뷰 - 생성 예 : ALIAS 사용

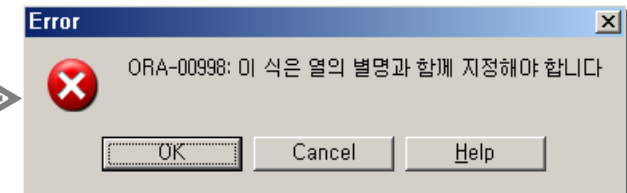
뷰 컬럼이 함수나 표현식에서 파생되는 경우 반드시 사용해야 함

```
CREATE OR REPLACE VIEW V_EMP ("Enm", "Gender", "Years") AS
SELECT EMP_NAME,
       DECODE(SUBSTR(EMP_NO, 8,1), '1', '남자', '3', '남자', '여자'),
       ROUND(MONTHS_BETWEEN(SYSDATE, HIRE_DATE)/12, 0)
FROM   EMPLOYEE;
```

COLUMN_NAME	DATA_TYPE	NULLABLE
Enm	VARCHAR2	N
Gender	VARCHAR2	Y
Years	NUMBER	Y

서브쿼리 부분에서 ALIAS 지정해도 됨

```
CREATE OR REPLACE VIEW V_EMP AS
SELECT EMP_NAME ,
       DECODE(SUBSTR(EMP_NO, 8,1), '1', '남자', '3', '남자', '여자'),
       ROUND(MONTHS_BETWEEN(SYSDATE, HIRE_DATE)/12, 0)
FROM   EMPLOYEE;
```





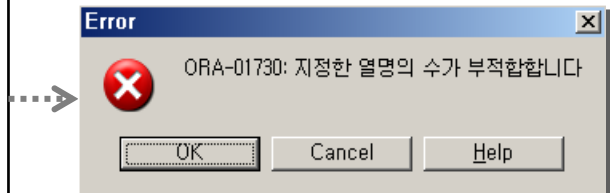
## 5.2.3 뷰 - 생성 예 : ALIAS 사용

특정 컬럼에만 선택적으로 ALIAS를 지정하는 것은 서브쿼리 부분에서만 가능

```
CREATE OR REPLACE VIEW V_EMP AS
SELECT EMP_NAME,
       DECODE(SUBSTR(EMP_NO, 8,1),
              '1','남자','3','남자','여자') AS "Gender",
       ROUND(MONTHS_BETWEEN(SYSDATE, HIRE_DATE)/12,0) AS "Years"
FROM   EMPLOYEE;
```

COLUMN_NAME	DATA_TYPE	NULLA
EMP_NAME	VARCHAR2	N
Gender	VARCHAR2	Y
Years	NUMBER	Y

```
CREATE OR REPLACE VIEW V_EMP ("Gender", "Years") AS
SELECT EMP_NAME ,
       DECODE(SUBSTR(EMP_NO, 8,1),'1','남자','3','남자','여자'),
       ROUND(MONTHS_BETWEEN(SYSDATE, HIRE_DATE)/12, 0)
FROM   EMPLOYEE;
```



서브쿼리의 EMP\_NAME 컬럼은 ALIAS 없이 그대로 사용하려는 의미로 생략  
 → 뷰 생성부분에서는 전체 컬럼에 대해 지정해야 함

### 5.2.3 뷰 - 생성 예 : 제약조건

- 뷰의 원래 목적은 아니지만 뷰를 통한 DML 작업은 가능함
- DML 작업 결과는 베이스 테이블의 데이터에 적용 → COMMIT/ROLLBACK 작업 필요
- 뷰를 통한 DML 작업은 여러 가지 제한이 있음(부록 참조)
- 뷰 생성 시 DML 작업에 대한 제한을 설정할 수 있음
  - WITH READ ONLY : 뷰를 통한 DML 작업 불가
  - WITH CHECK OPTION : 뷰를 통해 접근 가능한 데이터에 대해서만 DML 작업 수행 가능

## 5.2.3 뷰 - 생성 예 : 제약조건

WITH READ ONLY

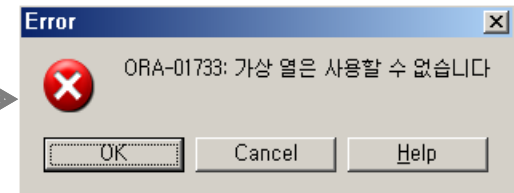
```
CREATE OR REPLACE VIEW V_EMP  
AS SELECT *  
   FROM EMPLOYEE  
WITH READ ONLY;
```

WITH READ ONLY를  
사용한 샘플 뷰

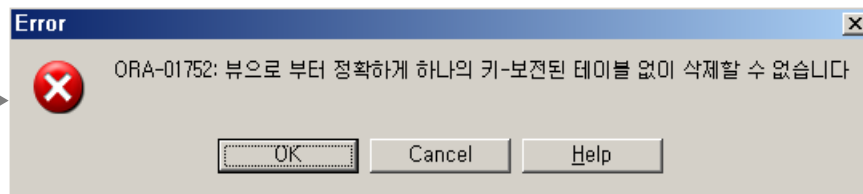
DML 작업에 따라 에러 유형은 다르지만 DML 작업을 허용하지 않는다

```
UPDATE V_EMP  
SET   PHONE = NULL;
```

```
INSERT INTO V_EMP (EMP_ID, EMP_NAME, EMP_NO)  
VALUES ('666', '오현정', '666666-6666666');
```



```
DELETE FROM V_EMP;
```



## 5.2.3 뷰 - 생성 예 : 제약조건

WITH CHECK OPTION - 조건에 따라 INSERT/UPDATE 작업 제한(DELETE는 제한 없음)

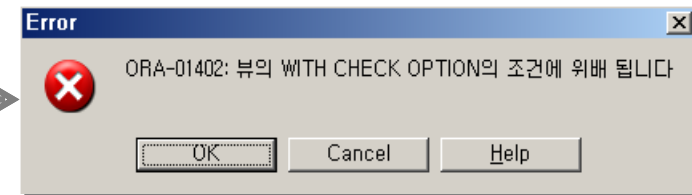
```
CREATE OR REPLACE VIEW V_EMP
AS SELECT EMP_ID, EMP_NAME, EMP_NO, MARRIAGE
FROM EMPLOYEE
WHERE MARRIAGE = 'N'
WITH CHECK OPTION;
```

EMP_ID	EMP_NAME	EMP_NO	MARRIAGE
124	정지현	641231-2269080	N
149	성해교	640524-2148639	N
205	임영애	790833-2105839	N

WITH CHECK OPTION을 사용한 샘플 뷰

```
INSERT INTO V_EMP (EMP_ID, EMP_NAME, EMP_NO, MARRIAGE)
VALUES ('666', '오현정', '666666-6666666', 'Y');
```

```
UPDATE V_EMP
SET MARRIAGE = 'Y';
```



```
UPDATE V_EMP
SET EMP_ID = '000'
WHERE EMP_ID = '124';
```

EMP_ID	EMP_NAME	EMP_NO	MARRIAGE
000	정지현	641231-2269080	N
149	성해교	640524-2148639	N
205	임영애	790833-2105839	N

뷰를 생성할 때 사용한 WHERE 조건에 적용되지 않는 범위에서는 허용됨

## 5.2.4 뷰 - 내용 확인

뷰 생성 시 사용한 서브쿼리 자체가 데이터 디렉터리에서 저장됨

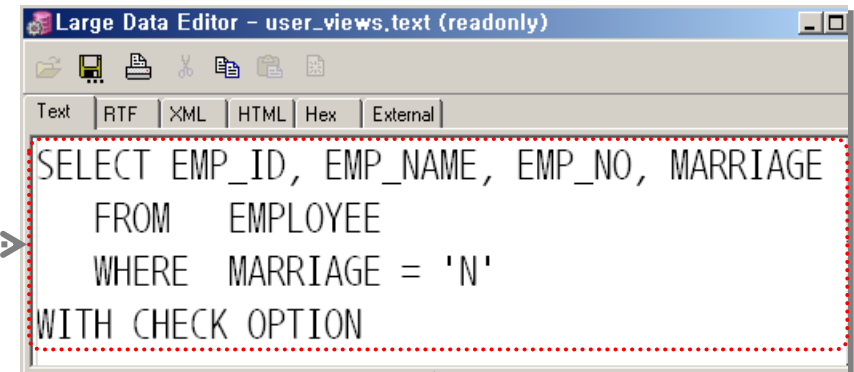
```
CREATE OR REPLACE VIEW V_EMP  
AS SELECT EMP_ID, EMP_NAME, EMP_NO, MARRIAGE  
FROM EMPLOYEE  
WHERE MARRIAGE = 'N'  
WITH CHECK OPTION;
```

```
SELECT VIEW_NAME, TEXT  
FROM USER_VIEWS  
WHERE VIEW_NAME = 'V_EMP';
```

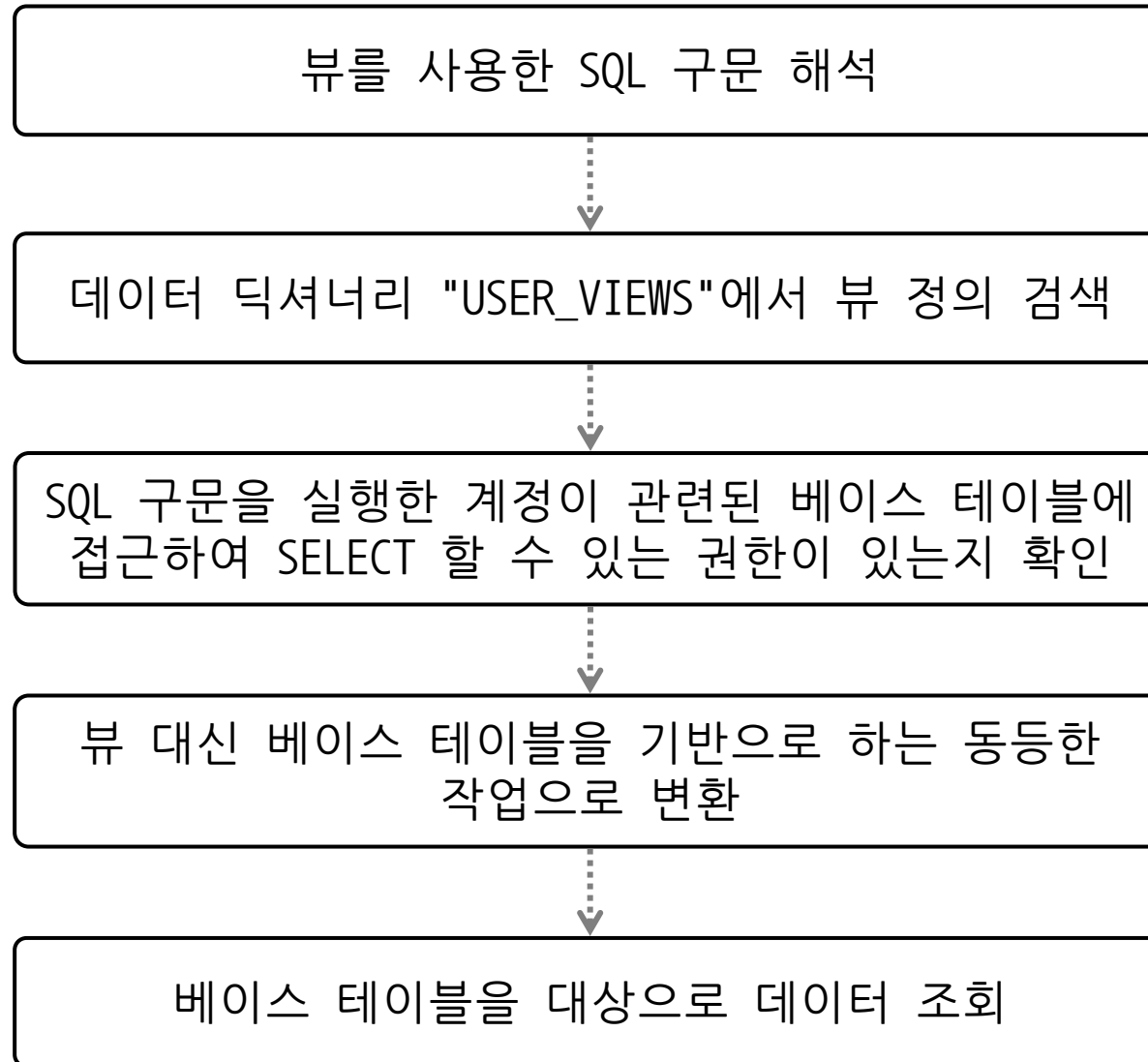
USER\_VIEWS

뷰 정보를 관리하는 데이터 디렉터리

VIEW_NAME	TEXT
V_EMP	<Long>



## 5.2.5 뷰 - 데이터 조회 절차



## 5.2.6 뷰 - 사용

```
CREATE OR REPLACE VIEW V_EMP_INFO
AS SELECT EMP_NAME, DEPT_NAME, JOB_TITLE
FROM   EMPLOYEE
LEFT JOIN DEPARTMENT USING (DEPT_ID)
LEFT JOIN JOB USING (JOB_ID);
```

V\_EMP\_INFO 데이터(일부)

EMP_NAME	DEPT_NAME	JOB_TITLE
한선기	해외영업3팀	대표이사
강중훈	해외영업3팀	부사장
최만식	해외영업3팀	부사장
안석규	기술지원팀	부장
조재형	기술지원팀	부장
김순이	해외영업1팀	부장
정도연	기술지원팀	차장

```
SELECT EMP_NAME
FROM   V_EMP_INFO
WHERE  DEPT_NAME = '해외영업1팀'
AND    JOB_TITLE = '사원';
```

EMP_NAME
정지현
성해교

## 5.2.6 뷰 - 사용

```
CREATE OR REPLACE VIEW V_DEPT_SAL ("Did", "Dnm", "Davg")
AS SELECT NVL(DEPT_ID, 'N/A'),
          NVL(DEPT_NAME, 'N/A'),
          ROUND(AVG(SALARY), -3)
FROM    DEPARTMENT
RIGHT JOIN  EMPLOYEE USING (DEPT_ID)
GROUP BY DEPT_ID, DEPT_NAME;
```

V\_DEPT\_SAL 데이터

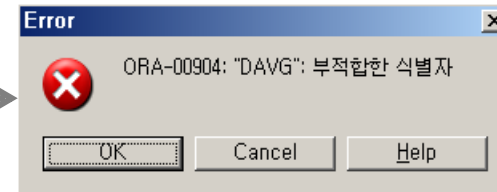
Did	Dnm	Davg
N/A	N/A	1900000
60	기술지원팀	3300000
90	해외영업3팀	6033000
10	본사 인사팀	2517000
80	해외영업2팀	2255000
50	해외영업1팀	2300000
20	회계팀	2500000

" " 를 사용하여  
alias를 지정한  
경우에는 " "까지  
기술해야 함

```
SELECT "Dnm", "Davg"
FROM    V_DEPT_SAL
WHERE   "Davg" > 3000000;
```

Dnm	Davg
기술지원팀	3300000
해외영업3팀	6033000

```
SELECT Dnm, Davg
FROM    V_DEPT_SAL
WHERE   Davg > 3000000;
```



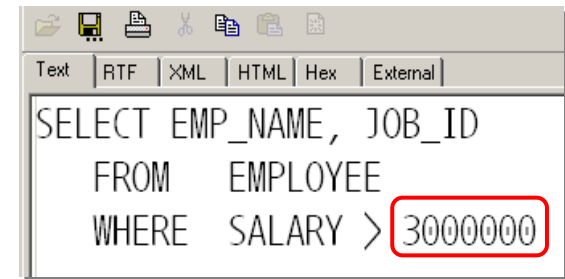


## 5.2.7 뷰 - 수정

뷰 수정 의미 → 별도 구문 없음

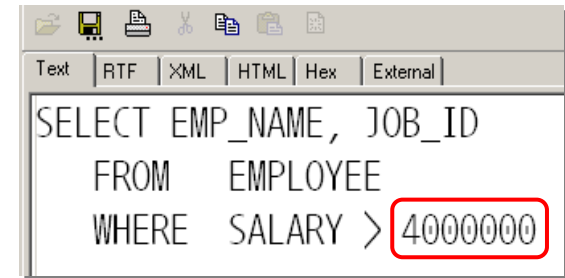
- 뷰를 삭제하고 새로 생성
- 기존 내용을 덮어써서 수정

```
CREATE OR REPLACE VIEW V_EMP
AS SELECT EMP_NAME, JOB_ID
FROM EMPLOYEE
WHERE SALARY > 3000000;
```



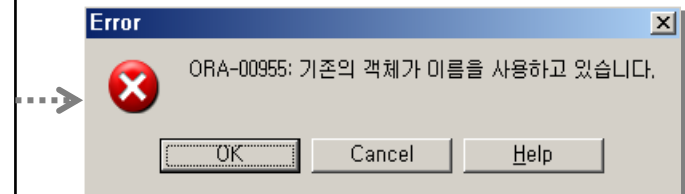
CREATE OR REPLACE 구문을 사용했으므로 기존에 존재하는 V\_EMP 이름을 그대로 사용하고, 내용만 수정되었음

```
CREATE OR REPLACE VIEW V_EMP
AS SELECT EMP_NAME, JOB_ID
FROM EMPLOYEE
WHERE SALARY > 4000000;
```



CREATE 구문을 사용했으므로 이미 사용중인 V\_EMP 이름이 중복되어 에러 발생함

```
CREATE VIEW V_EMP
AS SELECT EMP_NAME, JOB_ID
FROM EMPLOYEE
WHERE SALARY > 4000000;
```



### 5.2.8 뷰 - 삭제

데이터 디렉터리에서 저장된 서브쿼리를 삭제하는 의미

```
DROP VIEW view_name ;
```

## 5.2.9 뷰 - 인라인 뷰 Inline View 개념

별칭을 사용하는 서브쿼리 → 일반적으로 FROM 절에서 사용

```
CREATE OR REPLACE VIEW V_DEPT_SALAVG ("Did", "Davg")
AS SELECT NVL(DEPT_ID, 'N/A'),
          ROUND(AVG(SALARY),-3)
FROM      EMPLOYEE
GROUP BY  DEPT_ID;

SELECT EMP_NAME, SALARY
FROM      EMPLOYEE
JOIN      V_DEPT_SALAVG ON ( NVL(DEPT_ID, 'N/A') = "Did" )
WHERE     SALARY > "Davg"
ORDER BY 2 DESC;
```

```
SELECT EMP_NAME, SALARY
FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
            ROUND(AVG(SALARY),-3) AS "Davg"
      FROM EMPLOYEE
      GROUP BY DEPT_ID) INLV
JOIN EMPLOYEE ON ( NVL(DEPT_ID, 'N/A') = INLV."Did" )
WHERE SALARY > INLV."Davg"
ORDER BY 2 DESC;
```

Did	Davg	EMP_NAME	SALARY
N/A	1900000	한선기	9000000
50	2300000	조재형	3800000
20	2500000	안석규	3500000
10	2517000	권상후	3410000
90	6033000	김순이	3400000
80	2255000	임영애	2640000
60	3300000	박하일	2600000
		엄정하	2420000
		심하균	2300000

- FROM 절이 수행되면서 별칭 INLV로 지정된 뷰가 생성되고 사용됨
- 별도로 생성하는 경우와 동일한 효과

### 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석 개념


- Top N 분석 : 조건에 맞는 최상위(또는 최하위) 레코드 N개를 식별해야 하는 경우에 사용
  - 최상위 소득자 3명
  - 최근 6개월 동안 가장 많이 팔린 제품 3가지
  - 실적이 가장 좋은 영업 사원 5명
- 오라클 환경에서 Top N 분석 원리
  - 원하는 순서대로 정렬
  - ROWNUM이라는 가상 컬럼을 이용하여 정렬 순서대로 순번 부여
  - 부여된 순번을 이용하여 필요한 수 만큼 식별

## 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석

ROWNUM 개념

SQL 구문 수행 후, Result set 각 행에 1부터 시작하는 일련의 숫자를 자동으로 할당한 가상 컬럼

```
SELECT ROWNUM, EMP_NAME, SALARY
FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
             ROUND(AVG(SALARY), -3) AS "Davg"
      FROM EMPLOYEE
      GROUP BY DEPT_ID) INLV
JOIN EMPLOYEE ON ( NVL(DEPT_ID, 'N/A') = INLV."Did")
WHERE SALARY > INLV."Davg";
```



ROWNUM	EMP_NAME	SALARY
1	한선기	9000000
2	안석규	3500000
3	조재형	3800000
4	김순이	3400000
5	엄정하	2420000
6	심하균	2300000
7	박하일	2600000
8	권상후	3410000
9	임영애	2640000

## 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석

## ROWNUM 특징

- WHERE 절이 실행되면서 순차적으로 할당됨
- 할당된 후에는 변경되지 않음

```
SELECT ROWNUM, EMP_NAME, SALARY
FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
             ROUND(AVG(SALARY), -3) AS "Davg"
      FROM EMPLOYEE
      GROUP BY DEPT_ID) INLV
JOIN EMPLOYEE ON (NVL(DEPT_ID, 'N/A') = INLV."Did")
WHERE SALARY > INLV."Davg"
ORDER BY 3 DESC;
```

ROWNUM	EMP_NAME	SALARY
1	한선기	9000000
3	조재형	3800000
2	안석규	3500000
8	권상후	3410000
4	김순이	3400000
9	임영애	2640000
7	박하일	2600000
5	엄정하	2420000
6	심하균	2300000

- WHERE 절이 수행되면서 조건을 만족시키는 행에 ROWNUM을 할당한 결과로 1차 Result set을 생성
- 1차 Result set에 대해 정렬을 수행하므로 정렬 순서대로 ROWNUM이 할당될 수 없음

### 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석

#### ROWNUM 사용

- ROWNUM 값으로 특정 행을 선택할 수 없음
- 단, Result set의 1<sup>st</sup> 행(ROWNUM = 1)은 선택 가능

```
SELECT ROWNUM, EMP_NAME, SALARY
FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
            ROUND(AVG(SALARY), -3) AS "Davg"
      FROM EMPLOYEE
     GROUP BY DEPT_ID) INLV
JOIN EMPLOYEE ON ( NVL(DEPT_ID, 'N/A') = INLV."Did")
WHERE SALARY > INLV."Davg"
AND ROWNUM = 3;
```

.....> ROWNUM | EMP\_NAME | SALARY |


- WHERE 절이 모두 수행되어야 ROWNUM이 할당됨
- 특정 ROWNUM 값이 할당되기 이전이므로 실행 되지만 원하는 결과를 만들 수 없음

## 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석

ROWNUM 사용

- ROWNUM 값으로 특정 행을 선택할 수 없음
- 단, Result set의 1<sup>st</sup> 행(ROWNUM = 1)은 선택 가능

```
SELECT ROWNUM, EMP_NAME, SALARY
FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
             ROUND(AVG(SALARY), -3) AS "Davg"
       FROM EMPLOYEE
       GROUP BY DEPT_ID) INLV
JOIN EMPLOYEE ON ( NVL(DEPT_ID, 'N/A') = INLV."Did")
WHERE SALARY > INLV."Davg"
AND ROWNUM = 1;
```



ROWNUM	EMP_NAME	SALARY
1	한선기	9000000



## 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석

## ROWNUM 사용

- ROWNUM 값을 이용하여 일정 범위에 해당하는 행만 선택할 수 있음
- N 순위보다 같거나 작은 범위만 식별 가능 : 예) 상위 5건 → ROWNUM ≤ 5

```
SELECT ROWNUM, EMP_NAME, SALARY
FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
             ROUND(AVG(SALARY), -3) AS "Davg"
      FROM EMPLOYEE
      GROUP BY DEPT_ID) INLV
JOIN EMPLOYEE ON ( NVL(DEPT_ID, 'N/A') = INLV."Did")
WHERE SALARY > INLV."Davg"
AND ROWNUM ≤ 5;
```



ROWNUM	EMP_NAME	SALARY
1	한선기	9000000
2	안석규	3500000
3	조재형	3800000
4	김순이	3400000
5	엄정하	2420000

- 지정한 범위에 포함되는 행 선택 가능
- 원하는 순서대로 정렬된 결과는 아님

원하는 순서대로  
정렬된 결과

ROWNUM	EMP_NAME	SALARY
1	한선기	9000000
3	조재형	3800000
2	안석규	3500000
8	권상후	3410000
4	김순이	3400000

### 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석 구문

- 순번을 활용하려면 ROWNUM이 할당되기 전에 미리 정렬을 해야 함
- 미리 정렬된 결과를 가지고 있도록 하기 위해 인라인 뷰를 사용

```
SELECT * | select_list
FROM   ( SELECT select_list
          FROM   table_name
          ...
          ORDER BY 기준 컬럼 )
WHERE  ROWNUM <= (또는 < ) N;
```

## 5.2.9 뷰 - 인라인 뷰 활용 : Top N 분석 사용 예

```
SELECT ROWNUM, EMP_NAME, SALARY
FROM ( ( SELECT EMP_NAME, SALARY
        FROM (SELECT NVL(DEPT_ID, 'N/A') AS "Did",
                     ROUND(AVG(SALARY), -3) AS "Davg"
                FROM EMPLOYEE
                GROUP BY DEPT_ID) INLV
        JOIN EMPLOYEE ON ( NVL(DEPT_ID, 'N/A') = INLV."Did")
        WHERE SALARY > INLV."Davg"
        ORDER BY 2 DESC )
WHERE ROWNUM <= 5;
```



ROWNUM	EMP_NAME	SALARY
1	한선기	9000000
2	조재형	3800000
3	안석규	3500000
4	권상후	3410000
5	김순이	3400000

인라인 뷰를 사용하여 부서 별 평균급여 값을 기준으로 정렬된 결과를 먼저 생성

# Ch 5.

---

## DDL(Data Definition Language)

5.1 | 테이블 Table

5.2 | 뷰 View

5.3 | 시퀀스 Sequence

5.4 | 인덱스 Index

### 5.3.1 시퀀스 개념 및 생성 구문

순차적으로 정수 값을 자동으로 생성하는 객체

```
CREATE SEQUENCE sequence_name  
[ INCREMENT BY N ] [ START WITH N ]  
[ { MAXVALUE N | NOMAXVALUE } ] [ { MINVALUE N | NOMINVALUE } ]  
[ { CYCLE | NOCYCLE } ] [ { CACHE N | NOCACHE } ] ;
```

[구문 설명]

- INCREMENT BY *N*  
시퀀스 번호 증가/감소 간격(*N*은 정수, 기본 값 1)
- START WITH *N*
  - 시퀀스 시작 번호(*N*은 정수, 기본 값 1)
- MAXVALUE/NOMAXVALUE, MINVALUE/NOMINVALUE
  - MAXVALUE *N* : 시퀀스의 최대 값 임의 지정(*N*은 정수)
  - NOMAXVALUE : 표현 가능한 최대 값(오름차순:10<sup>27</sup>, 내림차순:-1)까지 생성
  - MINVALUE *N* : 시퀀스의 최소 값 임의 지정(*N*은 정수)
  - NOMINVALUE : 표현 가능한 최소 값(오름차순:1, 내림차순:-10<sup>26</sup>)까지 생성
- CYCLE/NOCYCLE  
최대/최소 값 도달 시 반복 여부 결정
- CACHE/NOCACHE  
지정한 수량 만큼 미리 메모리에 생성 여부 결정(최소 값 2, 기본 값 20)

## 5.3.2 시퀀스 생성 예 1

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

- 초기값 : 300부터 시작
- 증가값 : 5씩 증가
- MAXVALUE 310 : 310까지 생성
- NOCYCLE : MAXVALUE(310)까지 생성 후 더 이상 생성 안됨
- NOCACHE : 미리 메모리에 생성하지 않음

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

NEXTVAL  
300

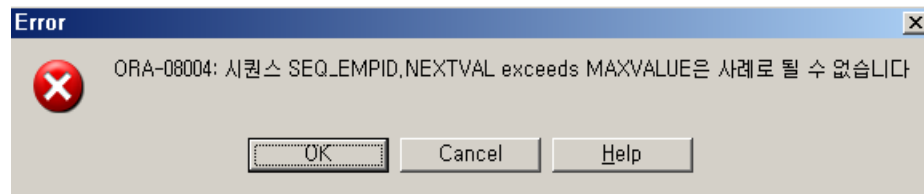
1회 사용

NEXTVAL  
305

2회 사용

NEXTVAL  
310

3회 사용



MAXVALUE 값에 도달했고 NOCYCLE이기 때문에 4회 사용시 에러 발생

## 5.3.2 시퀀스 생성 예 2

```
CREATE SEQUENCE SEQ2_EMPID  
START WITH 5  
INCREMENT BY 5  
MAXVALUE 15  
CYCLE  
NOCACHE;
```

- 초기값 : 5부터 시작
- 증가값 : 5씩 증가
- MAXVALUE 15 : 15까지 생성
- CYCLE : MAXVALUE(15) 까 지 생 성 후 1부터 5씩 증가하여 MAXVALUE 범위 안에서 반복 생성됨
- NOCACHE : 미리 메모리에 생성하지 않음

```
SELECT SEQ2_EMPID.NEXTVAL FROM DUAL;
```

NEXTVAL  
5

1회 사용

NEXTVAL  
10

2회 사용

NEXTVAL  
15

3회 사용

NEXTVAL  
1

4회 사용부터 1, 6, 11 이 반복적으로 생성됨

NEXTVAL  
6

NEXTVAL  
11

...

### 5.3.3 시퀀스 사용 방법

- NEXTVAL 속성

- 새로운 시퀀스 값을 반환
- '*sequence\_name*.NEXTVAL' 형태로 사용

- CURRVAL 속성

- 현재 시퀀스 값(NEXTVAL 속성에 의해 가장 마지막으로 반환된 시퀀스 값)을 반환
- '*sequence\_name*.CURRVAL' 형태로 사용
- NEXTVAL 속성이 먼저 실행되어야 사용 가능



## 5.3.3 시퀀스 사용 - NEXTVAL과 CURRVAL 관계

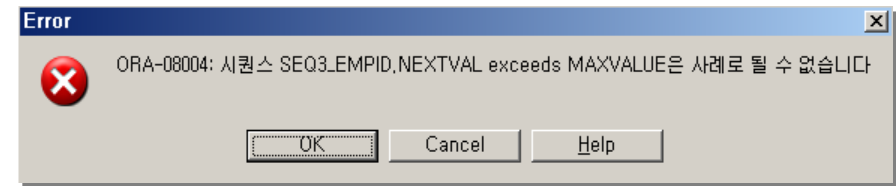
```
CREATE SEQUENCE SEQ3_EMPID
INCREMENT BY 5
START WITH 300 MAXVALUE 310
NOCYCLE NOCACHE;
```

```
SELECT SEQ3_EMPID.NEXTVAL FROM DUAL;
```

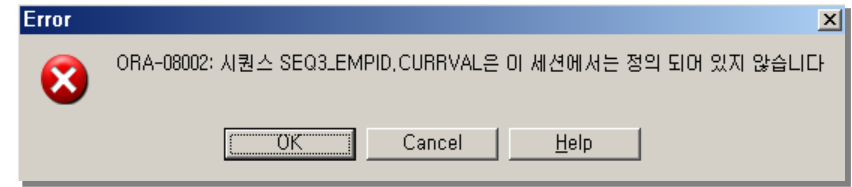
처음	2회	3회
NEXTVAL 300	NEXTVAL 305	NEXTVAL 310
	CURRVAL 300	CURRVAL 305

```
SELECT SEQ3_EMPID.CURRVAL FROM DUAL;
```

4회



CURRVAL 310	CURRVAL 310	...
----------------	----------------	-----



NEXTVAL 속성 사용 전에는 사용할 수 없음

## 5.3.3 시퀀스 사용

```
CREATE SEQUENCE SEQID  
INCREMENT BY 1  
START WITH 300  
MAXVALUE 310  
NOCYCLE NOCACHE;
```

```
INSERT INTO EMPLOYEE (EMP_ID, EMP_NO, EMP_NAME)  
VALUES (TO_CHAR(SEQID.NEXTVAL),  
        '850130-1558215', '김영민');  
  
INSERT INTO EMPLOYEE (EMP_ID, EMP_NO, EMP_NAME)  
VALUES (TO_CHAR(SEQID.NEXTVAL),  
        '840221-1361299', '구진표');  
  
SELECT EMP_ID, EMP_NO, EMP_NAME  
FROM   EMPLOYEE  
ORDER BY 1 DESC;
```

EMP_ID	EMP_NO	EMP_NAME
301	840221-1361299	구진표
300	850130-1558215	김영민
210	700813-1766819	감우섭
208	790411-1452247	이중기
207	640226-1358242	김술오

시퀀스 값을 사용하려는 EMP\_ID 컬럼 타입이 CHAR 타입이기 때문에  
TO\_CHAR 함수를 사용하여 타입 변환을 하였음

## 5.3.4 시퀀스 수정 및 삭제

- START WITH 값은 수정 불가
  - 시작 값을 변경하려면 삭제 후 새로 생성
  - 한번도 사용하지 않은 경우에도 수정 불가
- 변경된 값은 이후 시퀀스부터 적용
- 시퀀스 수정 구문

```
ALTER SEQUENCE sequence_name  
[ INCREMENT BY N ]  
[ { MAXVALUE N | NOMAXVALUE } ] [ { MINVALUE N | NOMINVALUE } ]  
[ { CYCLE | NOCYCLE } ] [ { CACHE N | NOCACHE } ] ;
```

- 시퀀스 삭제 구문

```
DROP SEQUENCE sequence_name;
```

## 5.3.4 시퀀스 수정 예

```
CREATE SEQUENCE SEQID2  
INCREMENT BY 1  
START WITH 300  
MAXVALUE 310  
NOCYCLE NOCACHE;  
  
SELECT SEQID2.NEXTVAL FROM DUAL;  
  
SELECT SEQID2.NEXTVAL FROM DUAL;  
  
ALTER SEQUENCE SEQID2  
INCREMENT BY 5;  
  
SELECT SEQID2.NEXTVAL FROM DUAL;
```

NEXTVAL
300

NEXTVAL
301

NEXTVAL
302



NEXTVAL
306

## 5.3.5 시퀀스 정보 확인

```
CREATE SEQUENCE SEQ1
INCREMENT BY 1
START WITH 1
NOCACHE;
SELECT SEQ1.NEXTVAL FROM DUAL;
SELECT SEQ1.CURRVAL FROM DUAL;
```

```
CREATE SEQUENCE SEQ2
INCREMENT BY 1
START WITH 1
CACHE 5;
SELECT SEQ2.NEXTVAL FROM DUAL;
SELECT SEQ2.CURRVAL FROM DUAL;
```

NEXTVAL	CURRVAL
1	1

```
SELECT SEQUENCE_NAME,
       CACHE_SIZE,
       LAST_NUMBER
FROM   USER_SEQUENCES
WHERE  SEQUENCE_NAME IN ('SEQ1','SEQ2');
```

SEQUENCE_NAME	CACHE_SIZE	LAST_NUMBER
SEQ1	0	2
SEQ2	5	6

LAST\_NUMBER

- CACHE 미사용 : 새로 반환될 시퀀스 값
- CACHE 사용 : CACHE로 생성된 이후 시퀀스 값(메모리에 생성된 시퀀스도 사용된 것으로 간주함)

# Ch 5.

---

## DDL(Data Definition Language)

5.1 | 테이블 Table

5.2 | 뷰 View

5.3 | 시퀀스 Sequence

5.4 | **인덱스** Index

## 5.4.1 인덱스 개념

- 키워드와 해당 내용의 위치가 정렬된 상태로 구성됨
- 키워드를 이용해 원하는 내용을 빠르게 찾기 위한 목적으로 사용
- 데이터베이스에서 인덱스는 컬럼 값을 이용해 원하는 행을 빠르게 찾기 위한 목적으로 사용

USER\_TAB\_COLUMNS view, 3-182  
     COLS synonym, 2-147  
 USER\_TAB\_COMMENTS view, 3-182  
 USER\_TAB\_HISTOGRAMS view, 3-182  
 USER\_TAB\_MODIFICATIONS view, 3-183  
 USER\_TAB\_PARTITIONS view, 3-183  
 USER\_TAB\_PRIVS view, 3-183  
 USER\_TAB\_PRIVS\_MADE view, 3-183  
 USER\_TAB\_PRIVS\_RECD view, 3-183  
 USER\_TAB\_STATISTICS view, 3-183  
 USER\_TAB\_SUBPARTITIONS view, 3-184  
 USER\_TABLES view, 3-184  
     TABS synonym, 3-154  
 USER\_TABLESPACES view, 3-184  
 USER\_TRANSFORMATIONS view, 3-184  
 USER\_TRIGGER\_COLS view, 3-184  
 USER\_TRIGGERS view, 3-184  
 USER\_TS\_QUOTAS view, 3-184  
 USER\_TUNE\_MVIEW view, 3-184  
 USER\_TYPE\_ATTRS view, 2-185

UTLRP.SQL script, B-3  
 UTLSAMPL.SQL script, B-3  
 UTLSCN.SQL script, B-3  
 UTLTKPRF.SQL script, B-4  
 UTLU101I.SQL script, B-5  
 UTLU101S.SQL script, B-5  
 UTLVALID.SQL script, B-4  
 UTLXPLAN.SQL script, B-4

**키워드** UTLXPLAN.SQL script, B-4 **위치(페이지 번호)**

## V

V\$ACCESS view, 4-2  
 V\$ACTIVE\_INSTANCES view, 4-2  
 V\$ACTIVE\_SERVICES view, 4-3  
 V\$ACTIVE\_SESS\_POOL\_MTH view, 4-3  
 V\$ACTIVE\_SESSION\_HISTORY view, 4-3  
 V\$ALERT\_TYPES view, 4-5  
 V\$AQ view, 4-5  
 V\$ARCHIVE view, 4-5  
 V\$ARCHIVE\_DEST view, 4-6

5.4.2 인덱스 구조

정렬된 특정 컬럼 값<sup>key</sup>과 해당 컬럼 값이 포함된 행 위치<sup>Rowid</sup>로 구성

DEPT\_ID 컬럼에  
인덱스 생성

DEPARTMENT 테이블

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀
80	해외영업2팀
90	해외영업3팀
30	마케팅팀

인덱스

DEPT_ID	ROWID
10	AAACikAAEAAAAlAAB
20	AAACikAAEAAAAlAAA
30	AAACikAAEAAAAlAAG
50	AAACikAAEAAAAlAAC
60	AAACikAAEAAAAlAAD
80	AAACikAAEAAAAlAAE
90	AAACikAAEAAAAlAAF

DEPT\_ID 컬럼 값은  
정렬된 형태로 구성



### 5.4.3 인덱스 사용 시 고려 사항

- 인덱스 특징
  - 테이블과 연관되지만 독립적인 객체
  - 자동적으로 사용되고 관리됨
  - DISK I/O를 줄임으로써 검색 속도를 향상시킬 수 있음
- 인덱스를 사용하는 것이 효율적인 경우
  - WHERE 절이나 JOIN 조건에 주로 사용되는 컬럼
  - UNIQUE 속성의 컬럼이나 NULL이 많이 포함된 컬럼
  - 넓은 범위의 값이 포함된 컬럼
- 인덱스를 사용하지 않는 것이 더 효율적인 경우
  - 테이블이 작은 경우(데이터가 적은 경우)
  - 테이블 갱신이 자주 발생하는 경우
  - 다량의 데이터가 조회되는 경우

### 5.4.4 인덱스 유형

인덱스를 생성하는 대상 컬럼에 따라 Unique Index, Nonunique Index로 구분

#### ▪ Unique Index

- Unique Index가 생성된 컬럼에는 중복 값이 포함될 수 없음
- 오라클은 'PRIMARY KEY' 제약조건을 생성하면 자동으로 해당 컬럼에 Unique Index를 생성
- PRIMARY KEY를 이용하여 access 하는 경우 성능 향상 효과 있음

#### ▪ Nonunique Index

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

## 5.4.4 인덱스 유형

UNIQUE INDEX

EMP_ID	ROWID
100	AAACiqAAEAAAAdAAA
101	AAACiqAAEAAAAdAAB
102	AAACiqAAEAAAAdAAC
103	AAACiqAAEAAAAdAAD
104	AAACiqAAEAAAAdAAE
107	AAACiqAAEAAAAdAAF
124	AAACiqAAEAAAAdAAG
141	AAACiqAAEAAAAdAAH
143	AAACiqAAEAAAAdAAI
144	AAACiqAAEAAAAdAAJ
149	AAACiqAAEAAAAdAAK
174	AAACiqAAEAAAAdAAL
176	AAACiqAAEAAAAdAAM
178	AAACiqAAEAAAAdAAN
200	AAACiqAAEAAAAdAAO
201	AAACiqAAEAAAAdAAP
202	AAACiqAAEAAAAdAAQ
205	AAACiqAAEAAAAdAAR
206	AAACiqAAEAAAAdAAS
207	AAACiqAAEAAAAdAAT
208	AAACiqAAEAAAAdAAU
210	AAACiqAAEAAAAdAAV

EMP_ID	EMP_NAME	DEPT_ID
100	한선기	90
101	강중훈	90
102	최만식	90
103	정도연	60
104	안석규	60
107	조재형	60
124	정지현	50
141	김예수	50
143	나승원	50
144	김순이	50
149	성해교	50
174	전우성	80
176	엄정하	80
178	심하균	...
200	고승우	10
201	박하일	50
202	권상후	10
205	임영애	10
206	염정하	...
207	김술오	20
208	이중기	20
210	감우섭	20

NONUNIQUE INDEX

DEPT_ID	ROWID
10	AAACiqAAEAAAAdAAQ
10	AAACiqAAEAAAAdAAR
10	AAACiqAAEAAAAdAAO
20	AAACiqAAEAAAAdAAT
20	AAACiqAAEAAAAdAAU
20	AAACiqAAEAAAAdAAV
50	AAACiqAAEAAAAdAAJ
50	AAACiqAAEAAAAdAAI
50	AAACiqAAEAAAAdAAH
50	AAACiqAAEAAAAdAAG
50	AAACiqAAEAAAAdAAK
50	AAACiqAAEAAAAdAAP
60	AAACiqAAEAAAAdAAD
60	AAACiqAAEAAAAdAAF
60	AAACiqAAEAAAAdAAE
80	AAACiqAAEAAAAdAAL
80	AAACiqAAEAAAAdAAM
90	AAACiqAAEAAAAdAAC
90	AAACiqAAEAAAAdAAB

## 5.4.5 인덱스 생성 1

[기본 구문]

```
CREATE [UNIQUE] INDEX index_name ON table_name ( column_list | function, expr );
```

Unique Index 생성

```
CREATE UNIQUE INDEX IDX_DNM ON DEPARTMENT(DEPT_NAME);
```

Nonunique Index 생성

```
CREATE INDEX IDX_JID ON EMPLOYEE (JOB_ID);
```

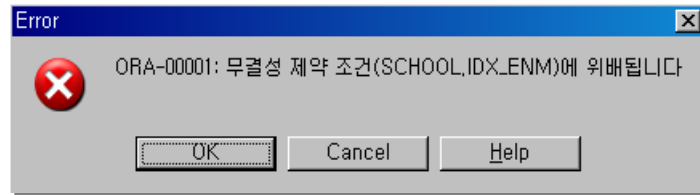
## 5.4.5 인덱스 생성 실습

1. EMPLOYEE 테이블의 EMP\_NAME 컬럼에 'IDX\_ENM' 이름의 Unique Index를 생성하시오.

```
CREATE UNIQUE INDEX IDX_ENM ON EMPLOYEE(EMP_NAME);
```

2. 다음과 같이 새로운 데이터를 입력해 보고, 오류 원인을 생각해 보시오.

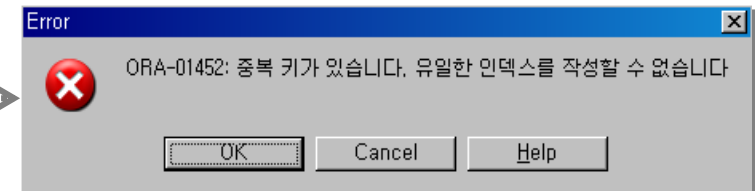
```
INSERT INTO EMPLOYEE (EMP_ID, EMP_NO, EMP_NAME)  
VALUES ('400', '871120-1243877', '감우섭');
```



EMP\_NAME 컬럼에 이미 '감우섭' 이름의 데이터가 존재하기 때문에 중복되는 값은 입력될 수 없음  
→ Unique Index는 UNIQUE 제약조건의 기능을 수행

3. EMPLOYEE 테이블의 DEPT\_ID 컬럼에 'IDX\_DID' 이름의 Unique Index를 생성해보고 오류 원인을 생각해 보시오.

```
CREATE UNIQUE INDEX IDX_DID ON EMPLOYEE(DEPT_ID);
```



### 5.4.6 인덱스 삭제


- 인덱스 삭제 구문을 이용하여 삭제
- 테이블이 삭제되면 관련된 인덱스는 함께 자동으로 삭제됨
- 인덱스 삭제 구문

```
DROP INDEX index_name ;
```

## 5.4.7 인덱스 정보 확인

EMPLOYEE 테이블에 생성된 인덱스 현황 조회

```
SELECT INDEX_NAME, COLUMN_NAME, INDEX_TYPE, UNIQUENESS  
FROM    USER_INDEXES  
JOIN    USER_IND_COLUMNS USING (INDEX_NAME, TABLE_NAME)  
WHERE   TABLE_NAME = 'EMPLOYEE';
```



INDEX_NAME	COLUMN_NAME	INDEX_TYPE	UNIQUENESS
PK_EMPID	EMP_ID	NORMAL	UNIQUE
UNI_EMPNO	EMP_NO	NORMAL	UNIQUE
IDX_ENM	EMP_NAME	NORMAL	UNIQUE
IDX_COMP	DEPT_ID	NORMAL	NONUNIQUE
IDX_COMP	JOB_ID	NORMAL	NONUNIQUE
IDX_HDATE	SYS_NC00013\$	FUNCTION-BASED NORMAL	NONUNIQUE

FUNCTION-BASED INDEX 경우 컬럼 이름이 다르게 표시됨

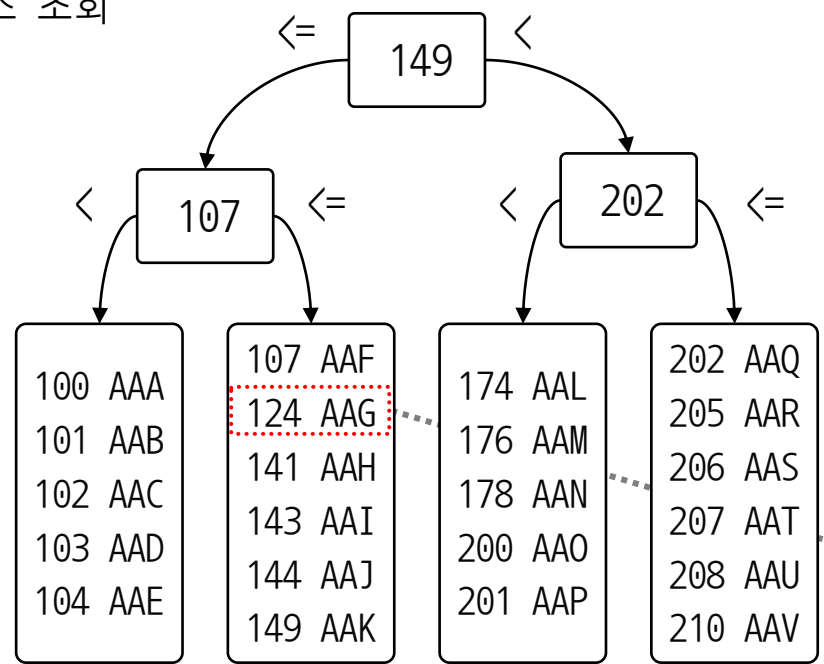
5.4.8 인덱스를 이용한 데이터 조회 방법

EMP\_ID 컬럼에 인덱스가 생성되어 있음

① QUERY 실행

```
SELECT EMP_NAME
FROM   EMPLOYEE
WHERE  EMP_ID = '124';
```

② 인덱스 조회



③ ROWID 식별

EMPLOYEE 테이블

ROWID	EMP_NAME
AAV	감우섭
AAB	강중훈
AAO	고승우
AAQ	권상후
AAJ	김순이
AAT	김술오
AAH	김예수
AAI	나승원
AAP	박하일
AAK	성해교
AAN	심하균
AAE	안석규
AAM	엄정하
AAS	염정하
AAU	이중기
AAR	임영애
AAL	전우성
AAD	정도연
AAG	정지현
AAF	조재형
AAC	최만식
AAA	한선기

④ ROWID를 이용하여 테이블 Access