

SQL 기초 정리



구조화 되어 있지 않은 데이터

우르모브 선수와 김두현 선수는 **K-리그**에서 선수로 활동하고 있습니다.

우르모브 선수는 아이콘스에서 **DF**로 활동하고 있으며 **1977년 8월 30일**에 태어났습니다.

키는 **180cm**에 몸무게는 **70kg**이고 등번호는 **4번**입니다.

김두현 선수는 삼성블루윙즈에서 선수생활을 하고 있으며 포지션은 **MF** 이며 생년월일은 **1982년 7월 14일** 생입니다.

키는 **175cm**에 몸무게는 **67kg** 이고 등번호는 우르모브 선수와 마찬가지로 **4번**입니다.

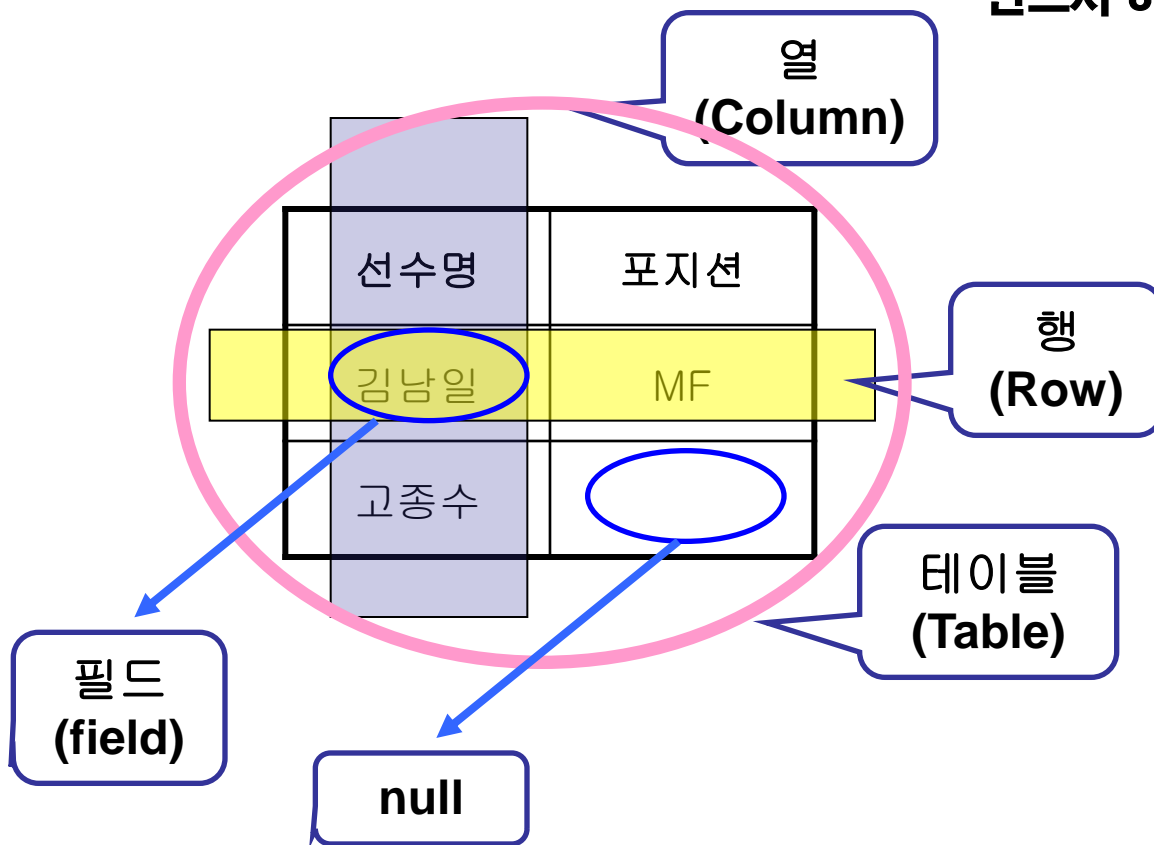
구조화된 데이터

선수명	소속팀	포지션	생년월일	신장	몸무게	등번호
우르모브	아이콘스	DF	1977/08/30	180	70	4
김두현	삼성블루윙즈	MF	1982/07/14	175	67	4

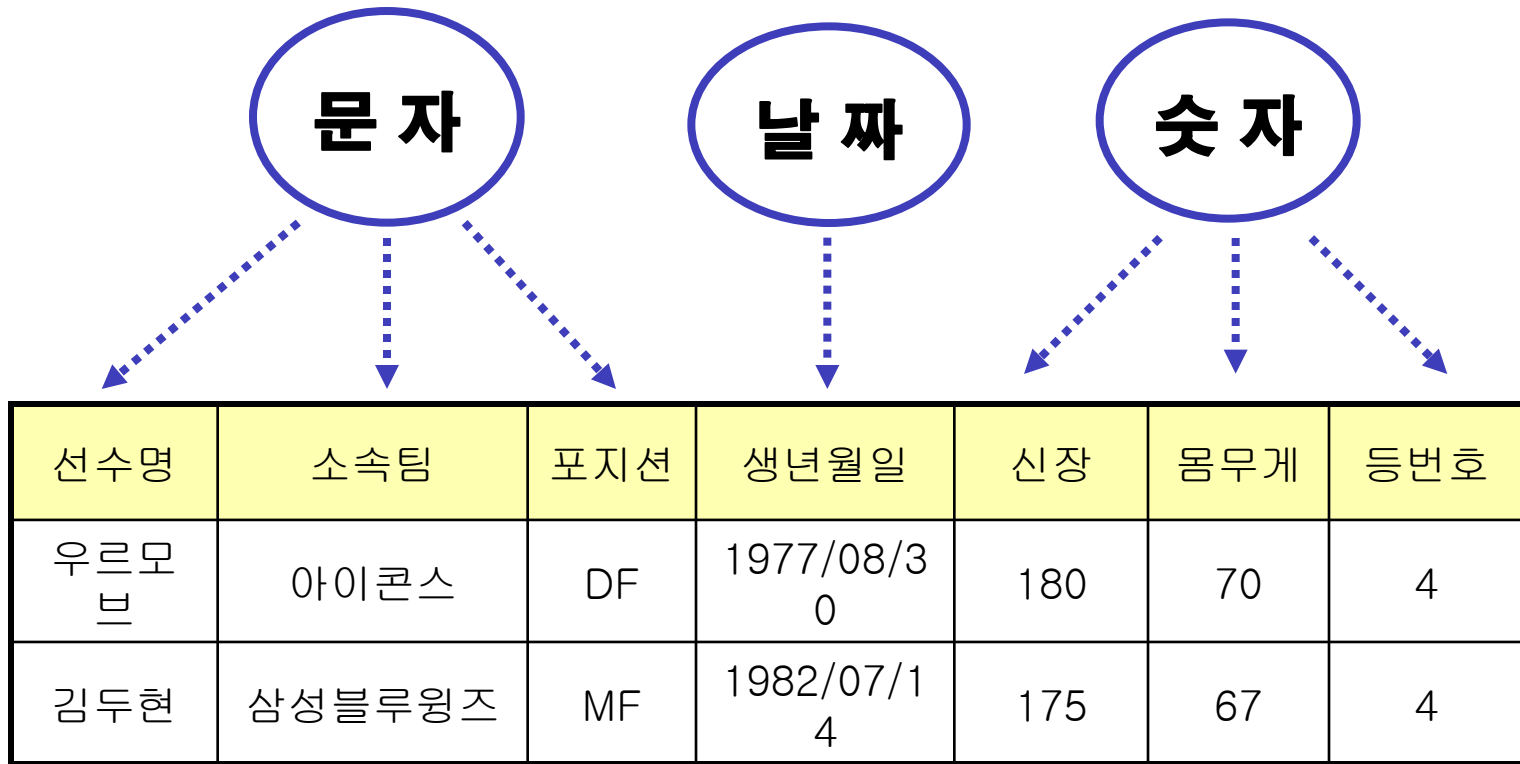
데이터 효과적 관리

테이블과 관련된 용어

Column과 Row를 가진 2차원 구조
반드시 하나 이상의 컬럼을 가진다.



SQL 데이터

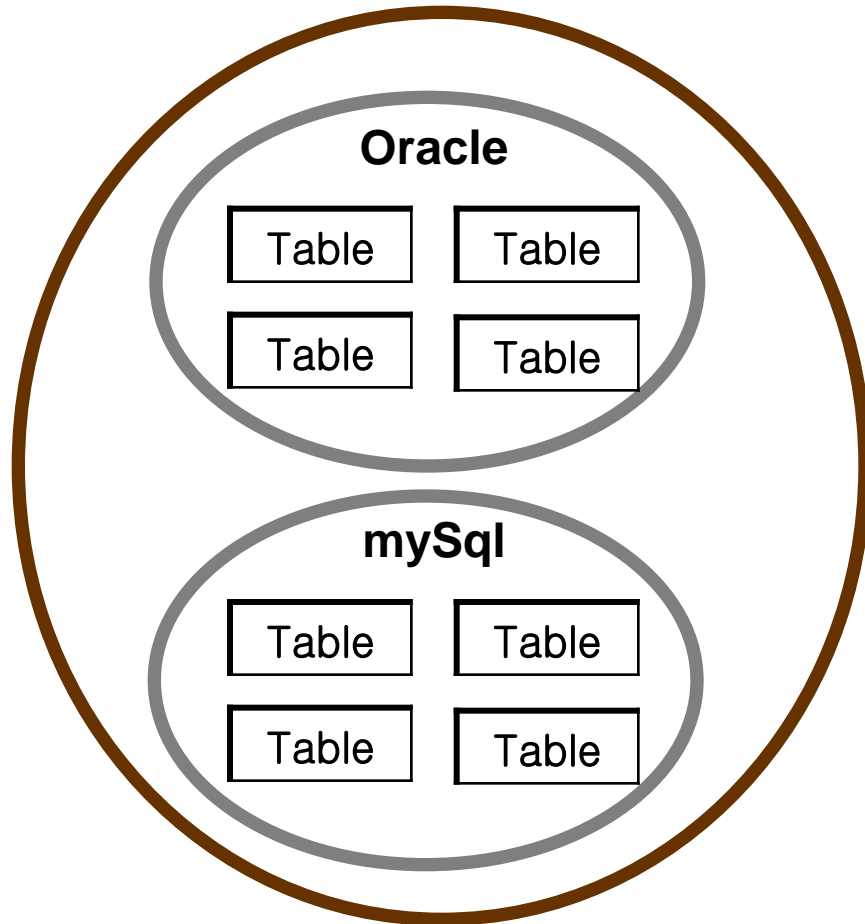


문자와 숫자를 구분하는 기준

→ 산술연산 가능 여부

전체 구조 생각하기

database 서버



접속요청

Toad

클라이언트

접속요청

sqlGate

SQL 이란?

- **SQL**은 데이터베이스에서 데이터 조작과 데이터 정의를 위해서 사용하는 언어
- 데이터 베이스와 대화한다는 것은 데이터베이스에 자료를 입력, 수정, 삭제, 조회를 할 수 있다는 의미이다.

용 어		설 명
입력	INSERT	데이터베이스에 관리하기를 원하는 자료를 저장
수정	UPDATE	데이터베이스에 입력되어 있는 자료들을 변경
삭제	DELETE	데이터베이스에 입력되어 있는 자료들을 삭제
조회	SELECT	데이터베이스에 입력되어 있는 자료들을 보려고 할 때



SQL 명령어와 관련된 용어

명령어의 종류	명령어
DQL (데이터 검색 조회)	SELECT
DML (데이터 조작어)	INSERT, DELETE, UPDATE
DDL (데이터 정의어)	CREATE, ALTER, RENAME DROP, TRUNCATE
TCL (트랜잭션 제어어)	COMMIT, ROLLBACK, SAVEPOINT

데이터 타입

데이터 타입	설 명
CHAR(S)	고정 길이 문자열 S : 1 - 2000
VARCHAR2(S)	가변 길이 문자열 S : 1 - 4000
NUMBER(P, S)	수치를 표현 P : 전체 숫자의 길이 S : 소수점 이하 표시
DATE	날짜와 시각을 표현

DDL - CREATE

1. 테이블 이름을 지정하고 각 칼럼들은 괄호 “()”로 묶어 지정한다.
2. 칼럼 뒤에 데이터 타입을 반드시 지정되어야 한다.
3. 각 칼럼들은 콤마 “,”로 구분되고, 항상 끝은 세미콜론 “;”으로 끝난다.
4. 한 테이블 안에서 칼럼 이름은 같을 수 없으며 다른 테이블에서의 칼럼 이름과는 같을 수 있다.

- =====
- 테이블 생성시 대.소문자 구분은 필요 없습니다.
 - **DATE** 형에는 별도의 크기를 지정하지 않습니다.
 - **VARCHAR2**형은 반드시 크기를 지정해야 합니다
 - **NUMBER, CHAR**는 크기를 정할 수 도 있고 정하지 않을 수도 있습니다.
 - 칼럼과 칼럼의 구분은 콤마(,)를 사용하며 마지막은 사용하지 않습니다.



DDL - CREATE

```
CREATE TABLE 테이블명(  
    칼럼명1 DATATYPE [DEFAULT 형식],  
    칼럼명2 DATATYPE [DEFAULT 형식],  
    . . . .  
);
```

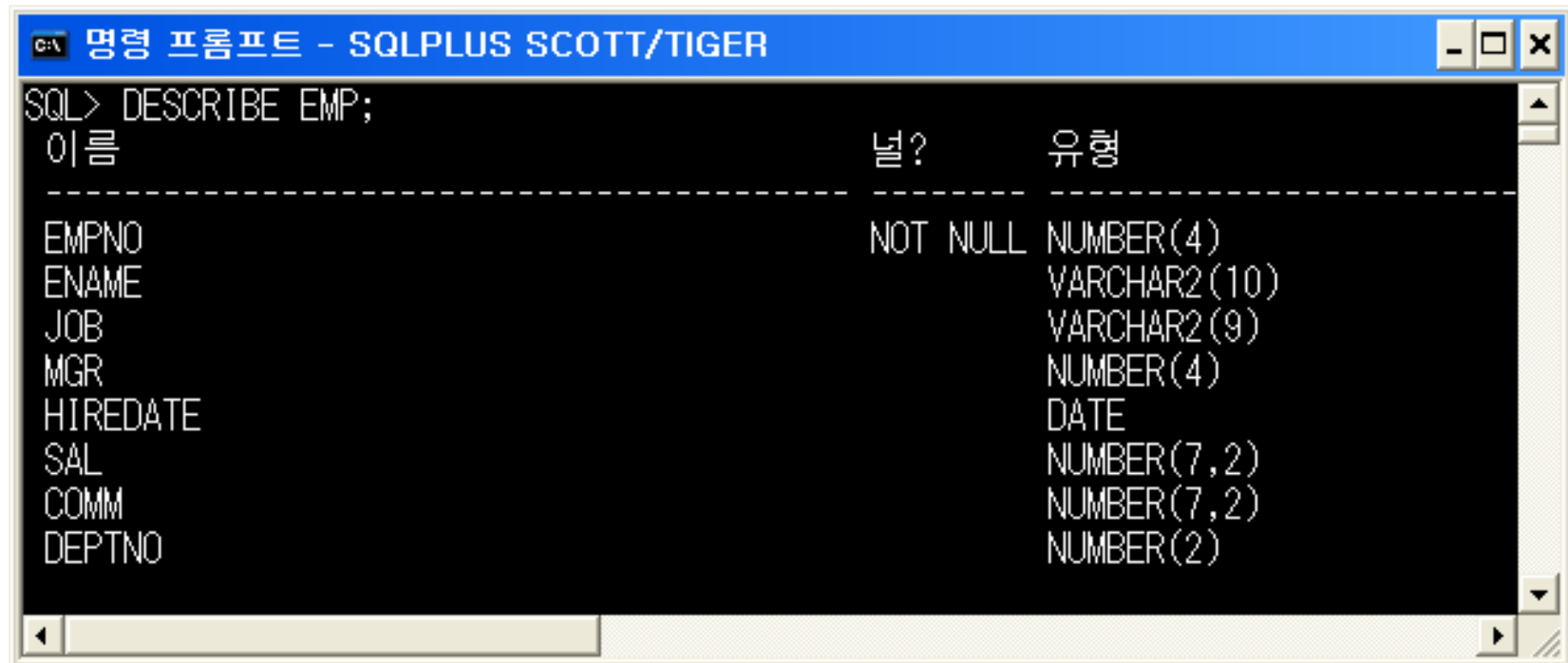
```
CREATE TABLE EX01(  
    EMPNO    VARCHAR2(10),  
    SAL      NUMBER(4) DEFAULT 0,  
    ENAME    VARCHAR2,    ➔ 오류발생  
    SEX      CHAR,        ➔ 가능  
    BIRTH    DATE         ➔ 가능  
);
```

DDL - DESCRIBE

DESCRIBE 테이블명;

DESC 테이블명;

DESCRIBE EMP; OR DESC EMP;



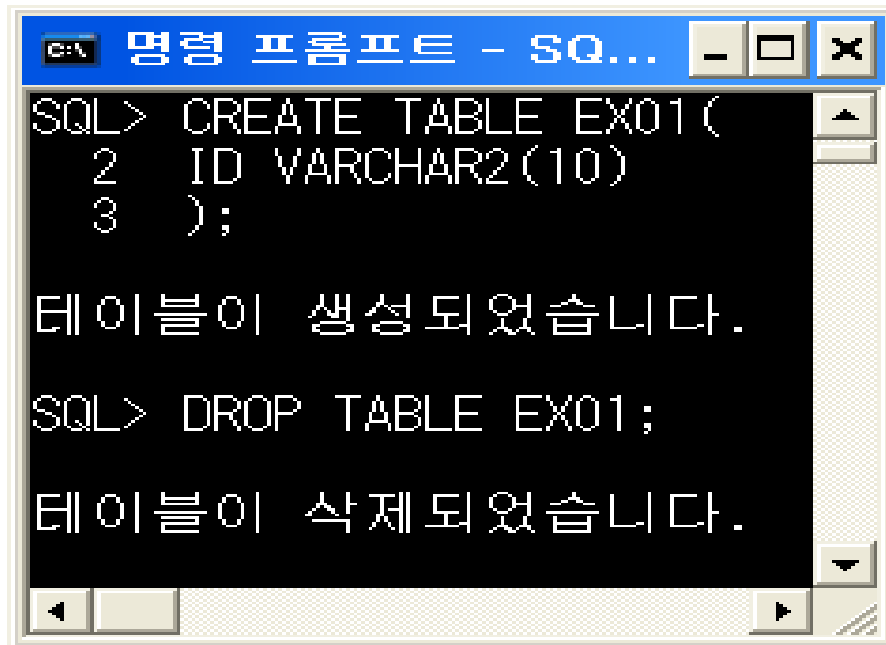
The screenshot shows a Windows-style window titled "명령 프롬프트 - SQLPLUS SCOTT/TIGER". The command prompt displays the command "SQL> DESCRIBE EMP;" and its output. The output is a table with three columns: "이름" (Name), "널?" (Null?), and "유형" (Type). The table lists the columns of the EMP table: EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, and DEPTNO, along with their respective data types and nullability.

이름	널?	유형
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

DDL - DROP

DROP

DROP TABLE 테이블명

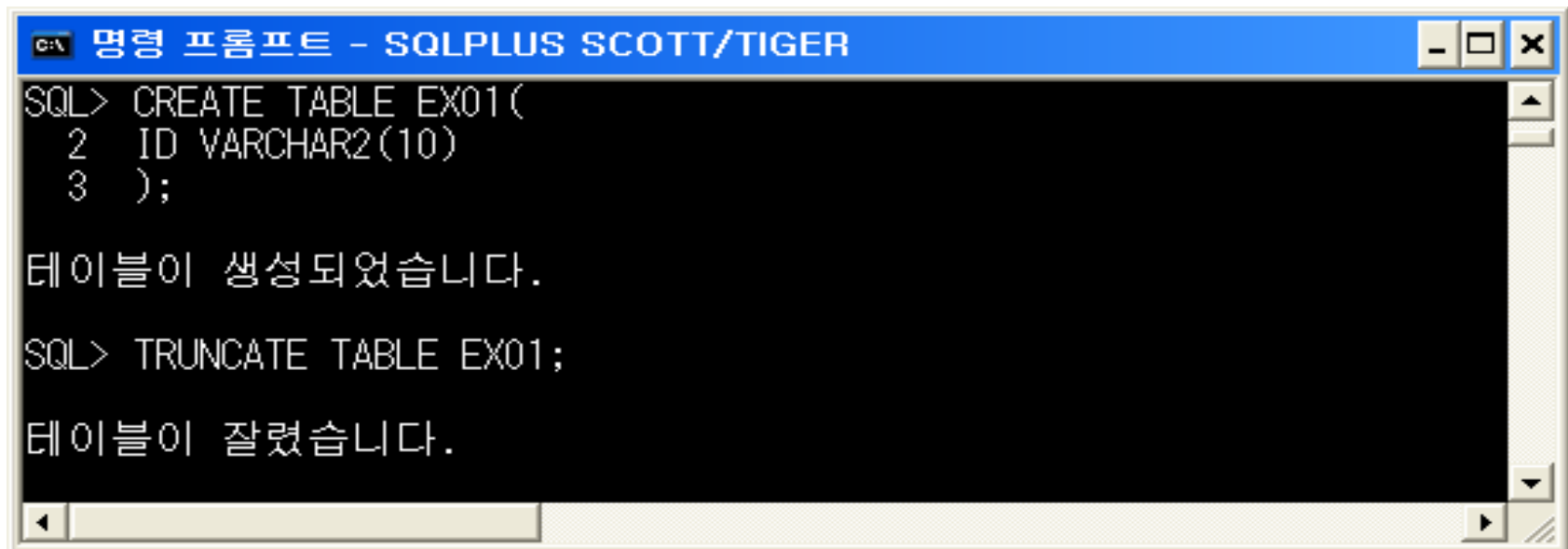


```
c:\ 명령 프롬프트 - SQ...  
SQL> CREATE TABLE EX01(  
2 ID VARCHAR2(10)  
3 );  
  
테이블이 생성되었습니다.  
  
SQL> DROP TABLE EX01;  
  
테이블이 삭제되었습니다.
```

DDL - TRUNCATE

TRUNCATE

TRUNCATE TABLE 테이블명;



```
C:\> 명령 프롬프트 - SQLPLUS SCOTT/TIGER
SQL> CREATE TABLE EX01(
  2  ID VARCHAR2(10)
  3  );

테이블이 생성되었습니다.

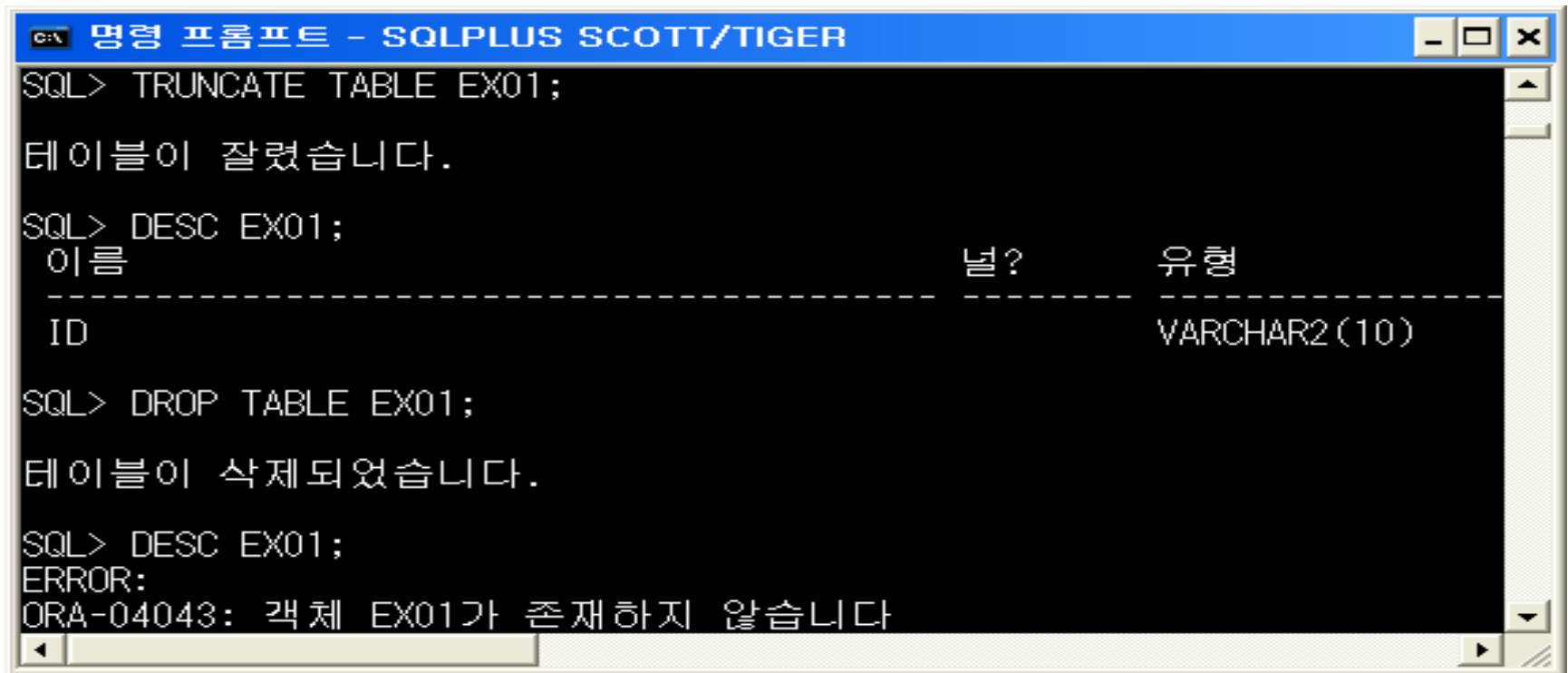
SQL> TRUNCATE TABLE EX01;

테이블이 잘렸습니다.
```

DROP와 TRUNCATE

DROP - 테이블의 구조와 데이터 모두 삭제

TRUNCATE - 테이블의 구조는 남겨두고 데이터만 삭제



```
C:\> 명령 프롬프트 - SQLPLUS SCOTT/TIGER
SQL> TRUNCATE TABLE EX01;
테이블이 잘렸습니다.

SQL> DESC EX01;
  이름                                널?       유형
-----
  ID                                V          VARCHAR2(10)

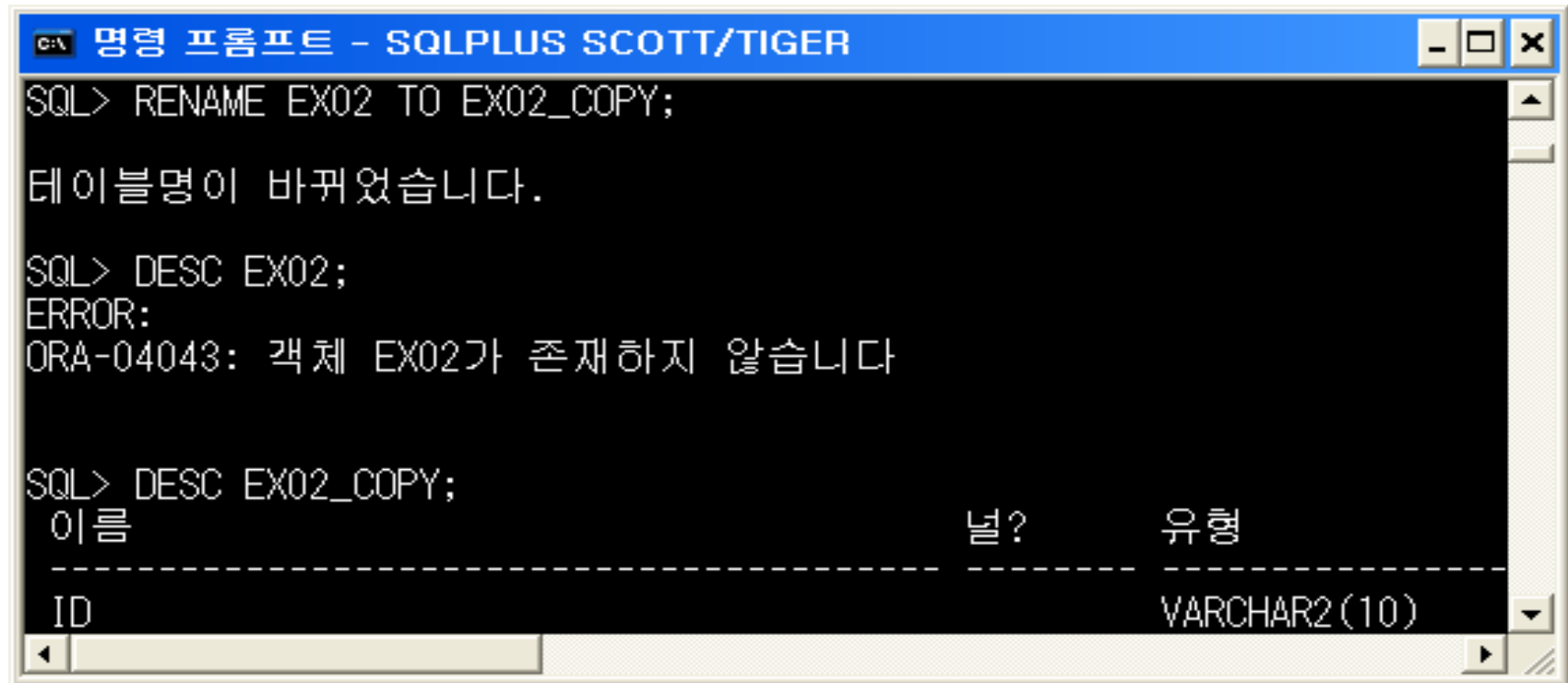
SQL> DROP TABLE EX01;
테이블이 삭제되었습니다.

SQL> DESC EX01;
ERROR:
ORA-04043: 객체 EX01가 존재하지 않습니다
```

DDL - RENAME

RENAME

RENAME 테이블명 **TO** 바꿀 테이블명



```
C:\> 명령 프롬프트 - SQLPLUS SCOTT/TIGER
SQL> RENAME EX02 TO EX02_COPY;

테이블명이 바뀌었습니다.

SQL> DESC EX02;
ERROR:
ORA-04043: 객체 EX02가 존재하지 않습니다

SQL> DESC EX02_COPY;
이름                                널?       유형
-----
ID                                V          VARCHAR2(10)
```


DML - 종류

- **INSERT** (새로운 데이터를 삽입)
- **UPDATE** (기존의 데이터를 변경)
- **DELETE** (기존의 데이터를 삭제)

INSERT INTO 테이블명(컬럼명,)
VALUES(값,);

UPDATE 테이블명
SET 컬럼명 = 변경할 값
[**WHERE** 조건] ;

DELETE
[**FROM**] 테이블명
[**WHERE** 조건] ;

INSERT - 1

INSERT INTO 테이블명(컬럼명,)

VALUES(값1,);

```
C:\ 명령 프롬프트 - sqlplus scott/tiger

SQL> DESC EX01;
이름                                널?       유형
-----
ID                                NOT NULL  NUMBER(10)
NAME                                VARCHA2(10)

SQL> INSERT INTO EX01(ID, NAME) VALUES(1, 'S');

1 개의 행이 만들어졌습니다.

SQL> SELECT * FROM EX01;

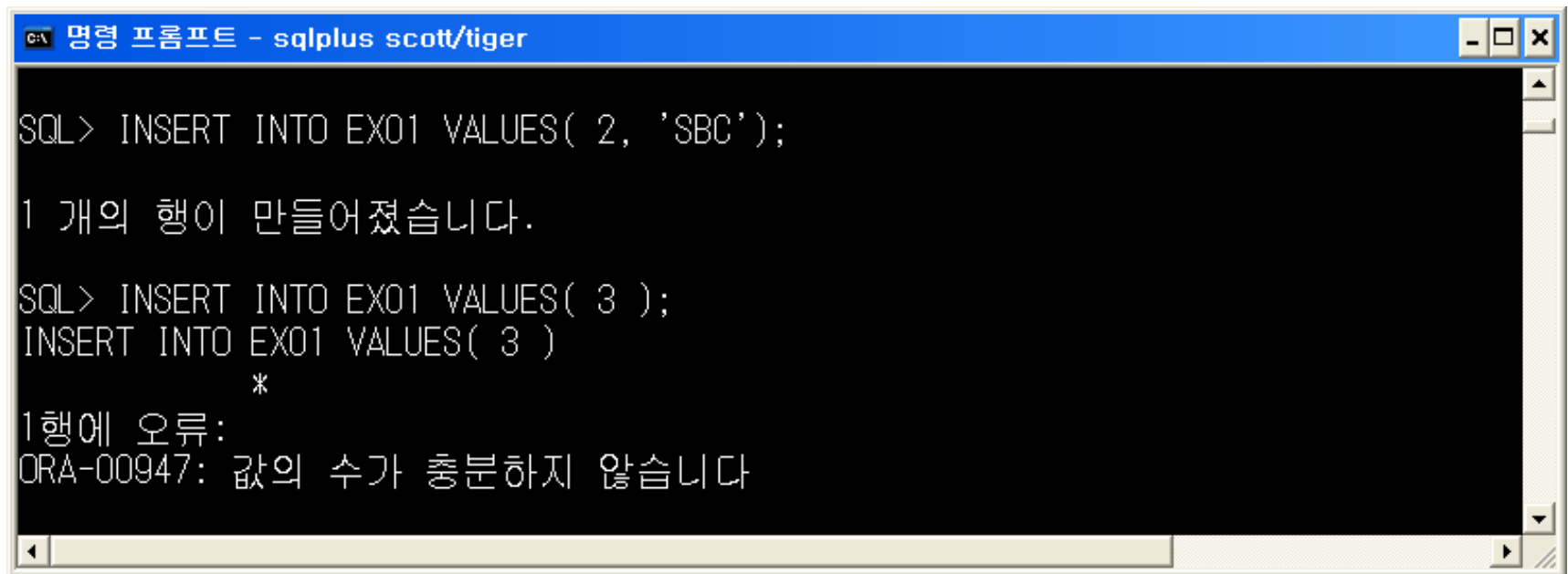
   ID NAME
----
    1  S

SQL>
```

INSERT - 2

INSERT INTO 테이블명 **VALUES**(값1,);

반드시 테이블의 컬럼의 수와 동일하게 값을 설정해야 한다.



```
C:\ 명령 프롬프트 - sqlplus scott/tiger

SQL> INSERT INTO EX01 VALUES( 2, 'SBC');

1 개의 행이 만들어졌습니다.

SQL> INSERT INTO EX01 VALUES( 3 );
INSERT INTO EX01 VALUES( 3 )
          *
1행에 오류:
ORA-00947: 값의 수가 충분하지 않습니다
```

UPDATE

UPDATE 테이블명 SET 컬럼명 = 바꿀값 WHERE 조건식

C:\ 명령 프롬프트 - sqlplus scott/tiger

```
SQL> SELECT * FROM EX01;
```

ID	NAME
1	S
2	SBC

```
SQL> UPDATE EX01 SET NAME = 'SON';
```

2 행이 갱신되었습니다.

```
SQL> SELECT * FROM EX01;
```

ID	NAME
1	SON
2	SON

DELETE

DELETE [FROM] 테이블명 WHERE 조건식

```
C:\ 명령 프롬프트 - sqlplus scott/tiger

SQL> SELECT * FROM EX01;

   ID NAME
-----
    1 SON
    2 SON

SQL> DELETE FROM EX01;

2 행이 삭제되었습니다.

SQL> SELECT * FROM EX01;

선택된 레코드가 없습니다.
```

SELECT

SELECT 조회할 컬럼명, 조회할 컬럼명

FROM 테이블명;

SELECT EMPNO, ENAME

FROM EMP;

1. “*”의 사용 - 테이블에 존재하는 모든 컬럼의 값을 보여준다.
2. ALIAS의 사용 - 화면에 보여지는 컬럼의 헤딩 부분을 바꾼다.
3. 산술 연산자 - 컬럼의 사칙 연산이 가능하다.
4. 합성 연산자 - 컬럼과 문자열의 합성이 가능하다.

ALIAS

ALIAS 의 사용

1. SELECT 컬럼명 AS ALIAS사용 : SELECT ID AS 아이디
2. SELECT 컬럼명 ALIAS사용 : SELECT ID 아이디
3. ALIAS에서 이중부호(" ")의 사용은 공백이나 대소문자 구분 시 사용한다

```
C:\ 명령 프롬프트 - sqlplus scott/tiger
SQL> SELECT * FROM EX01;

   ID NAME
-----
    1 S

SQL> SELECT ID 아이디, NAME AS 이름 FROM EX01;

아이디 이름
-----
    1 S
```



산술연산자

산술연산자의 사용

1. 종류 : +, -, *, /, ()



합성연산자

합성연산자의 사용

1. 2개의 수직 바(||)를 사용한다.
2. 칼럼과 문자 또는 다른 칼럼과 연결 시킨다.
3. 문자 표현식의 결과에 의해 새로운 칼럼의 결과를 보여준다.

WHERE - 조건

SELECT 컬럼명 [ALIAS]

FROM 테이블명

WHERE 조건식;

- **WHERE** 절에 사용되는 연산자

1) 비교 연산자

2) **SQL** 연산자

3) 논리 연산자

4) 조건의 부정



비교연산자

비교연산자의 사용

1. 조건을 부여하여 행들을 제한할 때 사용하는 연산자
2. =, >, <, >=, <=
3. CHAR, VARCHAR2와 같은 문자형 타입의 컬럼은 특정 값과 비교하기 위해서는 작은 따옴표(' ')로 묶어서 비교 처리 해야 합니다.



논리 연산자

논리 연산자의 사용

1. 여러 개의 조건들을 논리적으로 연결시키기 위한 연산자
2. AND : 앞,뒤의 조건이 모두 참일 경우만 참
3. OR : 앞, 뒤의 조건 중 하나만 참이면 참
4. NOT : 뒤에 오는 조건에 반대되는 결과를 돌려 줍니다.



논리 연산자

문1)

부서번호가 **90**번 또는 **100**번인 사원의 부서번호(**department_id**), 급여(**salary**),
사원번호(**employee_id**), 사원의 이름(**last_name**)을 출력하시오

문2)

부서번호가 **100**번인 사원 중에서 급여가 **10000** 이하인 사원의 정보를 출력
하시오

문3)

부서번호가 **90**번 또는 **100**번인 사원 중에서 급여가 **10000** 이하인 사원의
부서번호 (**department_id**), 급여(**salary**), 사원번호(**employee_id**), 사원의 이름
(**last_name**)을 출력하시오



SQL 연산자

SQL 비교연산자의 사용

1. BETWEEN a AND b : a와 b 사이의 값을 찾습니다.
2. IN(list) : list에 나열되는 값 중 하나만 일치되면 됩니다.
3. LIKE '비교문자열' : 비교 문자열과 형태가 일치하면 됩니다.
4. IS NULL : NULL값을 추출합니다.

SQL 연산자 – BETWEEN ~ AND

BETWEEN 의 사용

1. 컬럼명 BETWEEN a AND b : a가 b 보다 값이 작습니다.

– a(포함)와 b(포함) 사이의 값을 선택

예> salary between 1000 and 2000;

SQL 연산자 - IN

IN 의 사용

1. 컬럼명 IN (값1, 값2, ...)

예>

title in ('부장', '과장');

➔ (title = '부장' or title = '과장') 의미



SQL 연산자 - LIKE

LIKE 의 사용

1. % : 0개 이상의 어떤 문자를 의미합니다.

예> name like '김%';

2. _ : 1개인 단일 문자를 의미합니다.

예> name like '김__';



SQL 연산자 – IS NULL

IS NULL 의 사용

1. NULL 값과의 비교 연산은 거짓(FALSE)을 리턴
2. NULL 값과의 수치 연산은 NULL을 리턴
3. $\text{NULL} + 100 = \text{NULL}$, $\text{NULL} > 180 \Rightarrow \text{FALSE}$

부정연산자

구 분	연 산 자	연 산 자 의 의 미
부정 논리 연산자	!=	같지 않다
	^=	
	<>	
	NOT 컬럼명 =	
	NOT 컬럼명 > A	A 보다 크지 않다
SQL 비교 연산자	NOT BETWEEN A AND B	A와 B 사이에 포함되지 않는 값을 찾는다
	NOT IN (list)	리스트에 일치하지 않는 값을 찾는다
	IS NOT NULL	NULL이 아닌 것을 찾는다

ORDER BY - 정렬

SELECT [DISTINCT] 컬럼명 [ALIAS]

FROM 테이블명

ORDER BY 컬럼명 또는 표현식 [ASC or DESC]

=====

- **ASC** : 조회한 데이터를 오름차순으로 정렬한다(디폴트 값 생략 가능)
- **DESC** : 조회한 데이터를 내림차순으로 정렬한다.
- **ORDER BY** 절은 **SQL**문에서 마지막에 위치한다.
- 컬럼 **ALIAS**명의 사용이 가능하다.
- **NULL** 값은 오름차순일 경우 가장 마지막에 내림차순은 맨 처음 온다.
- 날짜형 데이터는 오름차순일 경우 가장 빠른 날이 먼저 출력된다.
- 기본적인 정렬방식은 오름차순이다.



GROUP BY

SELECT [DISTINCT] 컬럼명 [ALIAS]

FROM 테이블명

[**WHERE** 조건식]

GROUP BY 컬럼명 또는 표현식

- =====
- **SQL**문에서 일반적으로 **FROM**절 뒤에 오며 데이터들을 작은 그룹으로 분류하여 그룹에 대한 통계 정보를 얻을 때 사용한다.
 - 컬럼 **ALIAS** 명을 사용할 수 없다
 - 그룹 함수는 **WHERE**절에는 올 수 없다

- **COUNT(A)** : A행의 개수를 반환 합니다.
- **MAX(A)** : A행의 최대값을 반환 합니다.
- **AVG(A)** : A 행의 평균값을 반환 합니다.
- 그룹함수와 함께 **SELECT 절에 사용되는 컬럼은 GROUP BY 절에 명시해야만** 합니다.

```

C:\ 명령 프롬프트 - sqlplus scott/tiger

SQL> SELECT position 포지션, AVG(height) 평균키
      2 FROM player_t;
SELECT position 포지션, AVG(height) 평균키
      *
1행에 오류:
ORA-00937: 단일 그룹의 그룹 함수가 아닙니다
  
```

```
C:\ 명령 프롬프트 - sqlplus scott/tiger

1  SELECT position 포지션, AVG(height) 평균키
2  FROM player_t
3* GROUP BY 포지션
SQL> /
GROUP BY 포지션
      *
3행에 오류:
ORA-00904: 열명이 부적합합니다
```

**컬럼의 ALIAS
사용이 불가능**

```

C:\ 명령 프롬프트 - sqlplus scott/tiger
SQL> SELECT position 포지션, AVG(height) 평균키
      2 FROM player_t
      3 GROUP BY position;

포지션          평균키
-----
DF              180.677165
FW              180.041667
GK              186.547619
MF              176.408805
TC              178.833333
TD              176.5
TM              175
  
```


HAVING

SELECT [DISTINCT] 컬럼명 [ALIAS]

FROM 테이블명

[**WHERE** 조건식]

GROUP BY 컬럼명 또는 표현식

HAVING 그룹조건식

=====

- 그룹에 대한 제한 조건을 사용한다.

(**WHERE** 절 사용 불가)

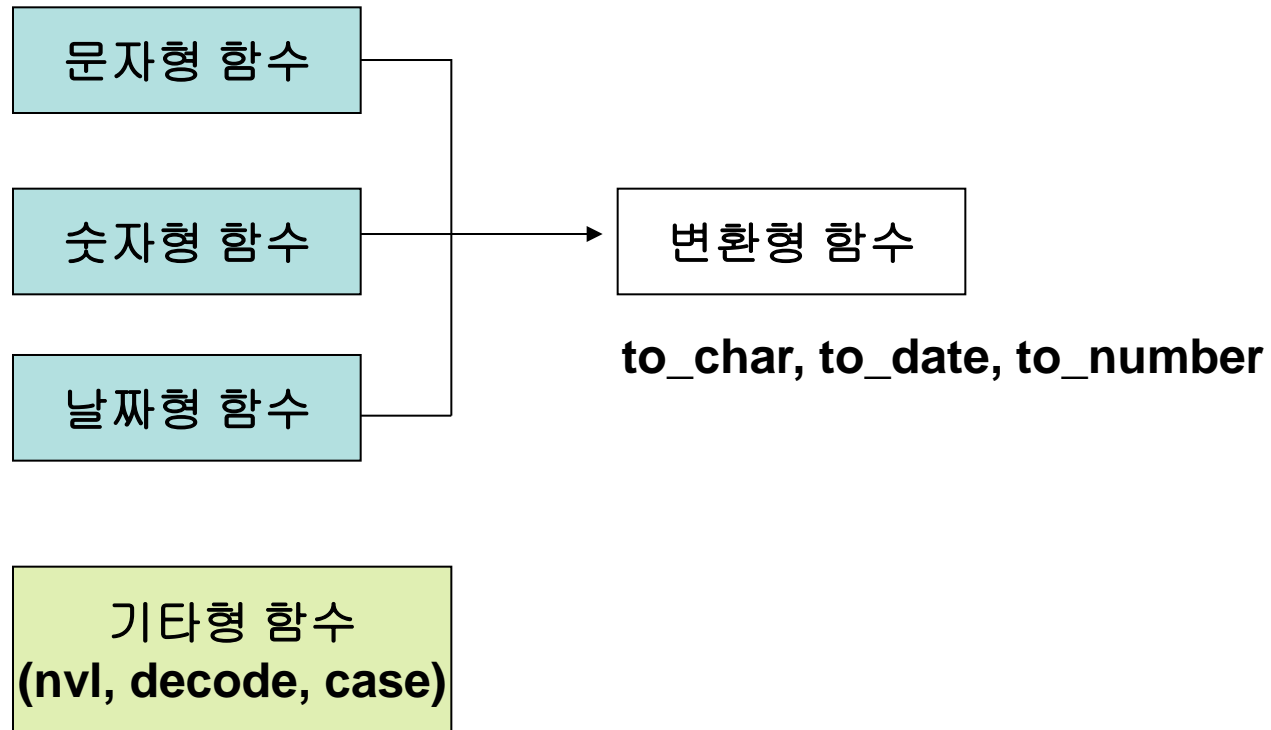
- 일반 적으로 **GROUP BY** 뒤에 사용

(**GROUP BY** 앞에서도 사용 가능)



내장함수

단일행 함수



문자형 함수

구분	문자형 함수	사용 목적
대소문자 전환함수	◎ LOWER(문자열)	문자열의 알파벳 문자를 소문자로 바꾼다.
	◎ UPPER(문자열)	문자열의 알파벳 문자를 대문자로 바꾼다.
	◎ INITCAP(문자열)	문자열에서 각 단어의 첫 글자를 대문자로, 나머지는 소문자로 만든다.
문자 조작함수	◎ CONCAT (문자열1, 문자열2)	문자열1과 문자열2을 연결한다. 연결연산자()와 동일하다.
	◎ SUBSTR (문자열, m[, n])	문자열 중 m위치에서 n개의 문자길이에 해당하는 문자를 돌려준다. n생략 시 끝까지이다.

문자형 함수

구분	문자형 함수	사용 목적
문자 조작 함수	◎ LENGTH(문자열)	문자열의 개수를 값으로 돌려준다.
	◎ LTRIM (문자열[, 지정문자])	문자열의 첫 문자부터 확인해서 지정문자가 나타나는 동안 해당문자를 제거한다.
	◎ RTRIM (문자열[, 지정문자])	문자열의 마지막 문자부터 확인해서 지정문자가 나타나는 동안 해당문자를 제거한다
	◎ TRIM ([leading trailing both] 지정문자 FROM 문자열)	문자열에서 머리말, 꼬리말, 또는 양쪽에 있는 지정문자를 제거한다. (leading trailing both가 생략되면 both값이 디폴트)
	◎ LPAD (문자열1, n [, 문자열2])	문자열1의 왼쪽에 문자열2를 덧붙여서 전체가 n개의 문자가 되도록 한다. 문자열2가 생략되면 공백으로 채워진다.

문자형 함수

구분	문자형 함수	사용 목적
문자 조작 함수	◎ RPAD (문자열1, n [, 문자열2])	문자열1의 오른쪽에 문자열2를 덧붙여서 전체가 n개의 문자가 되도록 한다. 문자열2가 생략되면 공백으로 채워진다.
	◎ TRANSLATE (문자열1, 문자열2, 문자열3)	문자열1 중에서 문자열2를 문자열3로 교체한다.
	◎ REPLACE (문자열1, 문자열2 [, 문자열3])	문자열1에서 문자열2를 찾아서 문자열3로 교체한다. 문자열3가 생략되면 문자열2가 제거된다.

숫자형 함수

숫자형 함수	사용 목적
◎ ABS(숫자)	숫자의 절대값을 돌려준다.
◎ MOD(숫자1, 숫자2)	숫자1을 숫자2로 나누어 나머지 값을 리턴.
◎ SIGN(숫자)	양수인지, 음수인지 0 인지 구별한다.
◎ CEIL(숫자)	숫자보다 크거나 같은 최소정수를 돌려준다.
◎ FLOOR(숫자)	숫자보다 작거나 같은 최대정수를 돌려준다
◎ ROUND(숫자 [, m])	숫자를 소수점 m자리에서 반올림하여 리턴. m이 생략되면 디폴트 값은 0 이다.
◎ TRUNC(숫자 [, m])	숫자를 소수 m자리에서 잘라서 버린다. m이 생략되면 디폴트 값은 0 이다.
◎ SIN, COS, TAN,...	숫자의 삼각함수 값을 돌려준다.


날짜형 함수

날짜형 함수	사용 목적
◎ SYSDATE	현재 날짜와 시각
◎ ADD_MONTHS(날짜, n)	날짜에 n개월을 더 한 날짜(n은 정수이어야 함)
◎ MONTHS_BETWEEN (날짜1, 날짜2)	날짜1과 날짜2 사이의 월수를 돌려주며 양수, 음수도 가능하다.
◎ LAST_DAY(날짜)	현재 날짜를 포함한 달의 마지막 날을 리턴.



다중행 함수

- 여러 행들의 그룹이 모여서 그룹당 단 하나의 결과를 돌려 주는 함수
- SELECT 및 HAVING절에 쓸 수 있다.
- SELECT 문장 상의 GROUP BY절은 행들을 그룹화 한다.
- HAVING절은 그룹을 제한하기 위해 사용한다.



– 여러 행들이 그룹이 모여서 그룹당 단 하나의 결과를 돌려주는 함수

그룹 함수	사용 목적
COUNT(*)	NULL 값을 포함한 행의 수를 출력.
COUNT(표현식)	표현식의 값이 NULL 값인 것을 제외한 행의 수를 출력.
SUM([DISTINCT ALL] 표현식)	표현식의 NULL 값을 제외한 합계를 출력.
AVG([DISTINCT ALL] 표현식)	표현식의 NULL 값을 제외한 평균을 출력.
MAX([DISTINCT ALL] 표현식)	표현식의 최대값을 출력. (문자, 날짜 데이터 타입도 사용가능)
MIN([DISTINCT ALL] 표현식)	표현식의 최소값을 출력. (문자, 날짜 데이터 타입도 사용가능)

변환형 함수

변환형 함수	사용 목적
◎ TO_CHAR (숫자 날짜 [, FORMAT])	숫자나 날짜를 주어진 FORMAT 형태대로 문자열 타입으로 변환한다.
◎ TO_NUMBER(문자열)	문자열을 숫자로 변환한다.
◎ TO_DATE (문자열 [, FORMAT])	문자열을 주어진 FORMAT 형태대로 날짜 타입으로 변환한다.

숫자 포맷

요소	설명	사용 예	사용 결과
9	숫자 위치를 나타낸다. '9'의 개수가 숫자를 출력할 자릿수 의미	'99999999'	1234
0	0을 출력한다.	'09999999'	0001234
.	표시한 위치에 소수점으로 출력한다.	'99999999.99'	1234.00
,	표시한 위치에 콤마(Comma)를 출력한다.	'9,999,999'	1,234

날짜 포맷

포맷 요소	설명	포맷 사용 결과
YY	년도	05
RR	다음 세기(Century)의 년도 마지막 두 자리 출력.	05
MM	월에 대한 두 자리 숫자(01~12)로 출력한다.	12
DD	월의 일수	24
HH, HH12 HH24	하루 중 시간 또는 시간(1~12) 시간(0~23)	5 17
MI	분(Minutes)를 숫자로 출력한다.(0~59)	33
SS	초(Seconds)를 숫자로 출력한다.(0~59)	59

to_char


1. to_char(숫자, '포맷') : 9 0 , .
2. to_char(날짜, '포맷') : y, mm, d, hh, mi, ss

```
SELECT SYSDATE, TO_CHAR(SYSDATE, 'yyyy-mm-dd')  
      , TO_CHAR(78000, '999,999')  
FROM dual;
```

SYSDATE	TO_CHAR(SYSDATE, 'YYYY-MM-DD')	TO_CHAR(78000, '999,999')
2013-05-06 오전 12:10:21	2013-05-06	78,000

기타 함수

함수명	설 명
NVL(표현식1, 표현식2)	표현식1이 NULL이면 표현식2의 값을 출력한다. 표현식1과 표현식2는 데이터 타입이 같아야 한다.
DECODE(표현식, 기준1, 값1, 기준2, 값2 디폴트값)	표현식의 값이 기준1이면 값1을 출력하고 기준2이면 값2를 출력한다. 그리고 기준 값이 없으면 디폴트 값을 출력
CASE WHEN 조건 THEN 처리할일 ELSE 디폴트값 END	WHEN 절의 조건이 참일때 THEN 절의 처리할일 을 실행 CASE 절의 끝은 반드시 END 로 표기함



조인 (join)



조인이란?

- **JOIN** 이란?

- 두 개 이상의 테이블 들을 연결해서 데이터를 조회 하는 것



종류

종 류	설 명
Equijoin	두 개의 테이블 간에 칼럼 값들이 정확하게 일치하는 경우에 사용한다. PK, FK의 관계를 기반으로 한다.
Non-Equijoin	두 개의 테이블 간에 칼럼 값들이 정확하게 일치하지 않는 경우에 사용
Outer Join	두 개의 테이블 간에 JOIN 을 걸었을 경우 JOIN 의 조건을 만족하지 않는 경우에도 그 데이터들을 보고자 하는 경우에 사용된다. (+)연산자를 사용한다.
Self Join	두 개의 테이블 같에 JOIN 을 거는 것이 아니라, 같은 테이블에 있는 행들을 JOIN 하는데 사용된다.



Cartesian Product

테이블들 간의 **JOIN** 조건을 생략하거나, 조건을 잘못 설정했을 경우 첫 번째 테이블의 모든 행들에 대해서 두 번째 테이블의 모든 행들이 **JOIN**이 되어 조회되는 상황이다.

```
SQL> SELECT e.first_name 이름, d.name 부서명  
       FROM employee e, department d;
```



Equi join

- 두 개의 테이블간에 칼럼 값들이 정확하게 일치하는 경우에 사용
- PK, FK의 관계를 기반으로 Equijoin가 성립하는 것이 일반적이다.
- ‘=’연산자에 의해서 성립된다.

```
SELECT 테이블명1.칼럼명, 테이블명2.칼럼명, ...  
FROM   테이블명1, 테이블명2  
WHERE  테이블명1.칼럼명1 = 테이블명2.칼럼명2;
```



Non-Equi join

- Non-Equijoin은 "=" 연산자가 아닌 다른 연산자를 사용하여 JOIN을 성립시킨다.

```
SQL> SELECT e.ename, e.job, e.sal, s.grade  
2  FROM emp e, salgrade s  
3  WHERE e.sal BETWEEN s.losal AND s.hisal;
```



Outer join


- JOIN 조건을 만족하지 못하는 경우에도 모든 행들을 다 보고자 하는 경우에 사용하는 JOIN으로서, '**(+)**' 연산자를 사용한다.
데이터가 모자른 쪽에 (+) 연산자를 사용한다.
= 의 앞 또는 뒤에 붙여서 사용한다.



Self join

- 두 개의 테이블간에 JOIN 조건을 적용하는 것이 아니라, 같은 테이블에 있는 행들을 가지고 다른 두 개의 테이블인 것처럼 JOIN 하는 방식

```
SQL> SELECT w.empno 사원번호, w.ename 사원명,  
2          m.ename 관리자명  
3 FROM emp w, emp m  
3 WHERE w.mgr = m.empno;
```



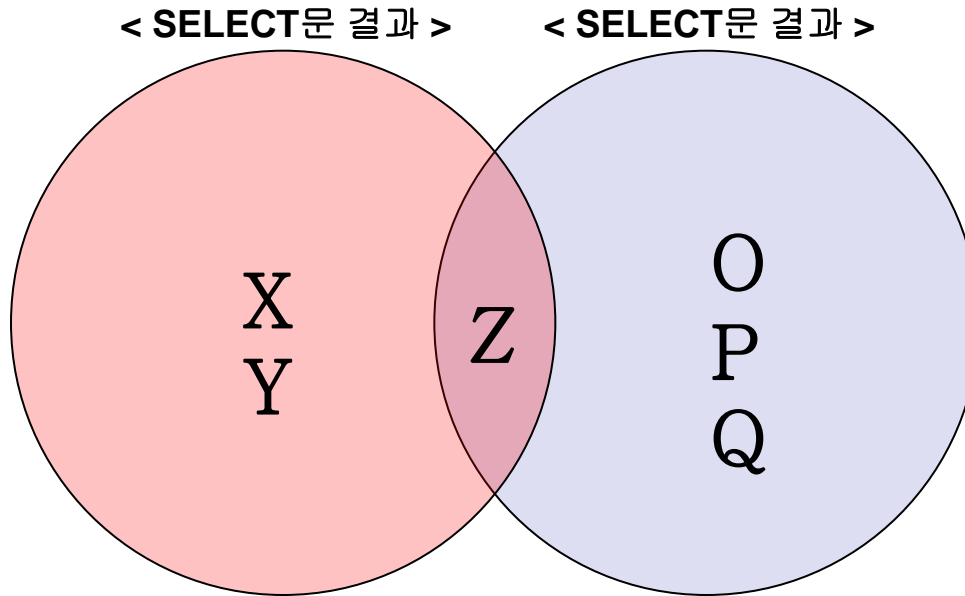
SET 연산자

의미 및 종류

- SET 연산자는 여러 개의 SQL문을 연결하여 조회한 데이터를 결합하는 방식을 사용한다.
- SET 연산자의 종류

종 류	설 명
UNION	여러 개의 SQL문의 결과에 대한 합집합. (자동 Sorting)
UNION ALL	여러 개의 SQL문의 결과에 대한 합집합과 공통부분을 더한 합집합
INTERSECT	여러 개의 SQL문의 결과에 대한 교집합
MINUS	앞의 SQL문의 결과에서 뒤의 SQL문의 결과의 차집합

결과




UNION ➔ X, Y, Z, O, P, Q
UNION ALL ➔ X, Y, Z, Z, O, P, Q
INTERSECT ➔ Z
MINUS ➔ X, Y



형태

```
SELECT 칼럼명1, 칼럼명2, ...  
FROM 테이블명1  
[ WHERE 조건식 ]  
[ GROUP BY 칼럼(Column)이나 표현식  
[ HAVING 그룹조건식 ] ]  
SET 연산자  
SELECT 칼럼명1', 칼럼명2', ...  
FROM 테이블명2  
[ WHERE 조건식 ]  
[ GROUP BY 칼럼(Column)이나 표현식  
[ HAVING 그룹조건식 ] ]  
[ ORDER BY 1, 2 [ ASC 또는 DESC ] ] ;
```



서브쿼리

(subquery)



SubQuery 의미

- 하나의 **SELECT** 문안에 포함되어 있는 또 다른 **SELECT** 문을 말하는 것.
- 알려지지 않은 기준에 의한 데이터 검색을 위해 사용한다.
- 괄호를 묶어서 사용한다.
- **Sub Query**가 사용 가능한 위치
 - 1) **WHERE** 절
 - 2) **HAVING** 절
 - 3) **INSERT** 문의 **INTO** 절
 - 4) **UPDATE** 문의 **SET** 절
 - 5) **SELECT** 또는 **DELETE** 문의 **FROM** 절

```
SELECT LAST_NAME, DEPARTMENT_ID, SALARY  
FROM EMPLOYEES  
WHERE DEPARTMENT_ID = ( SELECT DEPARTMENT_ID  
                        FROM EMPLOYEES  
                        WHERE LAST_NAME = 'KING' );
```

MAIN QEURY

SUB QEURY

동작방식에 따른 분류

- 동작하는 방식에 따른 SubQuery의 분류

종 류	설 명
Nested SubQuery	SubQuery가 MainQuery가 실행되기 전에 단 한번 수행된다. SubQuery의 결과 값이 Main Query에 제공되는 형태로 SubQuery가 Main Query의 제공자의 역할을 한다.
Correlated SubQuery	SubQuery 내에 Main Query의 칼럼(Column)들이 사용되므로 Main Query의 각 행에 대해 마지막 행에 도달할 때까지 SubQuery가 매번 실행된다. (심각한 오버헤드가 발생된다.)



Correlated SubQuery

- Main Query와 Sub Query가 서로 연관되어 있음
- 해석순서
 - Outer query의 한 Row를 얻는다.
 - 해당Row를 가지고 Inner Query를 계산한다.
 - 계산 결과를 이용 Outer query의 WHERE절을 실행
 - 결과가 참이면 해당 Row를 결과에 포함시킨다.

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary > (SELECT AVG(salary)
FROM employees
WHERE department_id = outer.department_id);
```


리턴 데이터에 다른 분류

종 류	설 명
Single Row SubQuery	SubQuery의 실행 결과인 단 하나의 행 이 MainQuery에 제공되는 형태로 단일행 비교연산자를 사용한다.
Multi Row SubQuery	SubQuery의 실행 결과인 여러 행 이 MainQuery에 제공되는 형태로 다중행 비교 연산자(IN, ANY, ALL)를 사용한다.
Multi Column SubQuery	SubQuery의 실행 결과가 여러 칼럼 이 반환되므로 MainQuery의 조건절에 여러 개의 칼럼을 동시에 비교하는 경우에 사용된다.



Single Row SubQuery

- SubQuery의 결과로 하나의 결과가 리턴되는 SubQuery
- 단일행 비교 연산자(=, <, <=, >, >=, <>) 들이 사용된다.

```
SELECT *  
  FROM EMPLOYEES  
 WHERE salary = (SELECT min(salary)  
                FROM employees);
```


```
SELECT *  
  FROM EMPLOYEES  
 WHERE salary < (SELECT avg(salary)  
                FROM employees);
```


Multi Column SubQuery

- SubQuery의 실행 결과에 여러 개의 칼럼(Column)이 반환되어 Main Query의 조건절에 사용되어 칼럼(Column) 들 전체를 동시에 비교하는 SubQuery

문제] 각 부서별로 최고급여를 받는 사원을 출력하시오.

```
SELECT department_id, employee_id, last_name, salary
FROM employees
WHERE (department_id, salary) IN
      (SELECT department_id, max(salary)
       FROM employees
       GROUP BY department_id);
```

시퀀스 (SEQUENCE)



SQL 객체 - 시퀀스

1. 오라클에서 제공하는 고유 숫자 생성 객체
2. `create sequence` 객체명;
3. 객체명.currVal : 마지막 생성된 고유번호
4. 객체명.nextVal : 고유번호 추출

시퀀스 생성 옵션

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

INCREMENT BY n: 번호 간격, 음수도 가능(기본설정 : 1)

START WITH n: 시작번호 (기본설정 : 1)

MAXVALUE n: 최대값(기본설정 : **NOMAXVALUE**)

MINVALUE n: 최소값 (기본설정 : 1)

NOCYCLE : 재사용 하지 않는다.(기본 설정)

CACHE n : 메모리에 올려서 사용할 개수. (기본설정 : **20개**)

시퀀스 생성

```
create sequence s1;  
select s1.nextVal from dual;  
select s1.currVal from dual;  
select s1.nextVal from dual;  
drop sequence s1;
```

```
-----  
create sequence s1  
increment by 1  
maxvalue 3;
```

```
select s1.nextVal from dual;  
select s1.nextVal from dual;  
select s1.nextVal from dual;  
select s1.nextVal from dual;  
-- error 확인  
drop sequence s1;
```

cycle 사용 시 **cache n**은 **1cycle**보다 작아야 한다.

```
-----  
create sequence s1  
increment by 1  
maxvalue 3  
cache 2  
cycle;
```

```
select s1.nextVal from dual;  
select s1.nextVal from dual;  
select s1.nextVal from dual;  
select s1.nextVal from dual;
```