

## ERD(Entity Relationship Diagram) 표기법

(ERD는 크게 IE표기법과 바커표기법으로 나뉨)

1. 엔터티 : 둘다 박스로 표현(바커의 경우 모서리가 둥근 사각형으로 표기하기도 함)

2. 관계(차수)

차수	IE	바커
1:1		
1:N		
N:M		

3. 관계(선택성)

선택성	IE(동그라미)	바커(점선)
필수		
선택		

\*\*바커 표기법은 점선의 방향이 중요함(바커는 자신 엔터티에 대한 표현을 반대쪽에 함)

4. 관계(식별/비식별)

관계	IE	바커
식별		
비식별		

\*\* 바커 표기법은 UID Bar(|)로 식별관계와 비식별관계를 구분

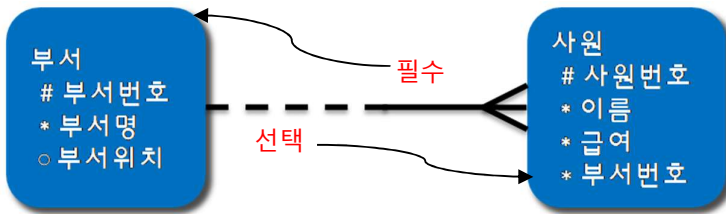
5. 식별자(주식별자)

관계	IE	바커
주식별자(PK)	맨위 네모칸 배치	#
일반속성	밑 네모칸 배치	* 또는 ○

6. 속성(널 허용여부)

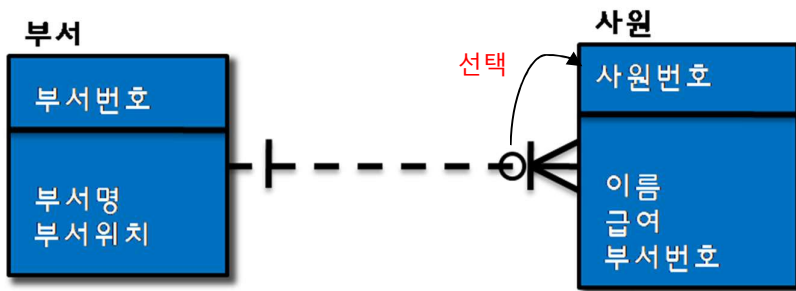
관계	IE	바커
널 허용	표기하지 않음	○
널 허용 X		*

예) 바커 표기법



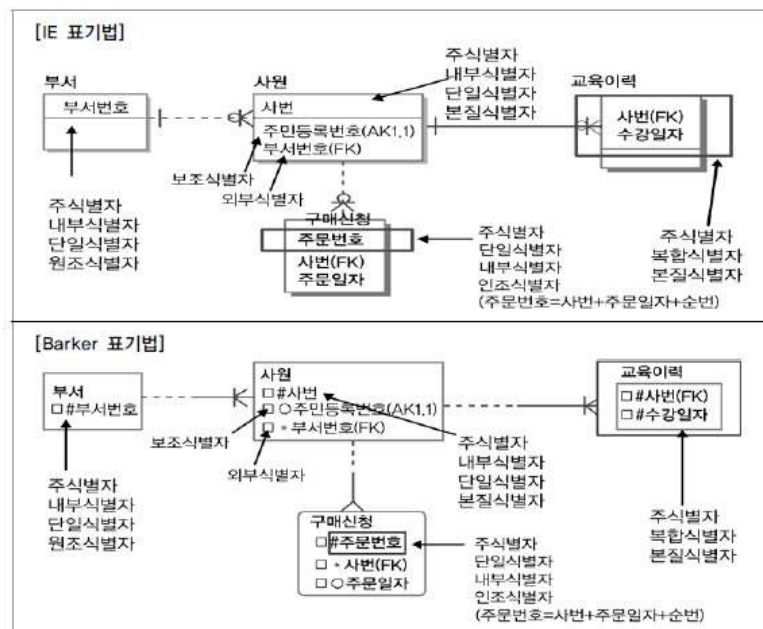
- ➔ 한 명 이상의 직원이 각 부서에서 일할 수 있음
- ➔ 부서는 필수적(사원이 있으면 부서도 있어야 함 -> 사원은 반드시 부서에 소속돼야 함)
- ➔ 사원은 선택적(부서는 있고 사원은 없어도 됨 -> 부서에는 사원이 없을 수 있음)
- ➔ UID Bar 가 없으므로 두 관계는 비식별관계
- ➔ 부서위치는 NULL 허용

예) IE 표기법



- ➔ 선택적 관계의 경우 IE 표기법에서는 동그라미로 표현(부서 필수, 사원 선택적)
- ➔ 비식별관계의 경우 IE 는 점선으로 표현

예) ERD 비교



[그림 1-1-42] 식별자의 분류-데이터 모델

## CHAR VS VARCHAR

### ● 데이터 입력의 차이

1. CHAR 타입은 고정 길이만큼 뒤에 공백을 채워 저장
2. VARCHAR 타입은 입력된 값 그대로 저장
3. 문자 타입 입력 시 제한된 길이보다 큰 "공백을 포함한 문자열" 값을 입력하면 고정 길이에 맞게 값에서 뒤 공백이 자르고 입력(ORACLE 은 입력 에러)

#### CASE1) CHAR 타입의 데이터 입력

The screenshot shows a SQL script in the 'SQL1 \*' window. The script creates a table named CHAR\_TEST1 with two columns: COL1 of type CHAR(5) and COL2 of type CHAR(7). It then inserts three rows of data. The first row inserts 'ABCDE' into both columns. The second row inserts 'abcde' into both columns. The third row inserts '12345' into both columns. The script ends with a SELECT statement to display the lengths of the columns.

```
1 CREATE TABLE CHAR_TEST1(  
2   COL1 CHAR(5),  
3   COL2 CHAR(7)  
4 );  
5  
6 INSERT INTO CHAR_TEST1 VALUES('ABCDE','ABCDE'); -- 입력가능  
7 INSERT INTO CHAR_TEST1 VALUES('abcde','abcde'); -- 입력가능  
8 INSERT INTO CHAR_TEST1 VALUES('12345','12345'); -- 입력가능  
9  
10 SELECT LENGTH(COL1), LENGTH(COL2) FROM CHAR_TEST1;
```

The 'Result' window shows the output of the SELECT statement. It has two columns: LENGTH(COL1) and LENGTH(COL2). The first row shows 5 and 7. The second row shows 5 and 7. The third row shows 5 and 7. The fourth row shows 5 and 7.

	LENGTH(COL1)	LENGTH(COL2)
1	5	7
2	5	7
3	5	7
4	5	7

☞ COL2의 값에 7 보다 작은 문자열을 삽입해도 모두 7 의 사이즈로 할당됨

#### CASE2) VARCHAR 타입의 데이터 입력

The screenshot shows a SQL script in the 'SQL1 \*' window. The script creates a table named VARCHAR\_TEST1 with two columns: COL1 of type VARCHAR2(5) and COL2 of type VARCHAR2(7). It then inserts three rows of data. The first row inserts 'ABCDE' into both columns. The second row inserts 'abcde' into both columns. The third row inserts '12345' into both columns. The script ends with a SELECT statement to display the lengths of the columns.

```
1 CREATE TABLE VARCHAR_TEST1(  
2   COL1 VARCHAR2(5),  
3   COL2 VARCHAR2(7)  
4 );  
5  
6 INSERT INTO VARCHAR_TEST1 VALUES('ABCDE','ABCDE'); -- 입력가능  
7 INSERT INTO VARCHAR_TEST1 VALUES('abcde','abcde'); -- 입력가능  
8 INSERT INTO VARCHAR_TEST1 VALUES('12345','12345'); -- 입력가능  
9  
10 SELECT LENGTH(COL1), LENGTH(COL2) FROM VARCHAR_TEST1;
```

The 'Result' window shows the output of the SELECT statement. It has two columns: LENGTH(COL1) and LENGTH(COL2). The first row shows 5 and 5. The second row shows 5 and 7. The third row shows 5 and 7.

	LENGTH(COL1)	LENGTH(COL2)
1	5	5
2	5	7
3	5	7

☞ 컬럼 사이즈와 상관없이 입력된 문자열의 크기 그대로 저장됨

## ● CHAR 타입 컬럼의 문자 상수 비교

### 1. 왼쪽에서부터 서로 다른 문자가 나올 때까지 비교

\* CHAR 컬럼끼리 서로 비교될 때 뒤 공백의 수만 다르고 값이 같다면 같은 것으로 간주!(뒤 공백 무시)

ex) 'ORACLE' = 'ORACLE ' -- 두 값은 서로 일치

\* 앞의 공백은 무시할 수 없음

ex) 'ORACLE' != ' ORACLE' -- 두 값은 서로 불일치(앞의 공백까지 같을 때만 같은 값으로 인정)

### 2. 달라진 첫 번째 문자 값에 따라 문자의 크기를 결정

\* 왼쪽부터의 값이 같을 때까지 체크 후 크기가 더 큰 쪽이 더 큰 값이 됨

ex) '가' < '가나' < '가나다'

\* 일반적으로 공백 < 특수기호 < 숫자 < 영어 < 한글의 크기 순서(DBMS 나 캐릭터셋 설정에 따라 다를 수 있음)

\* 대소문자를 구분하는 경우는 대문자 < 소문자 순서

## CASE1) 문자 값의 정렬

SQL1 *	
1	SELECT COL1
2	FROM VARCHAR_TEST1
3	ORDER BY COL1;
4	

Result	
	Grid Result Server Output Text Output Explain Plan
COL1	
1	ABCD
2	!@##\$
3	12345
4	ABCDE
5	abcde
6	가나

## CASE2) CHAR 타입끼리 비교 시 뒤 공백 무시, 앞 공백 중요

SQL1 *	
1	CREATE TABLE CHAR_TEST2(
2	COL1 CHAR(10),
3	COL2 CHAR(10)
4	);
5	INSERT INTO CHAR_TEST2 VALUES('ABC','ABC');
6	INSERT INTO CHAR_TEST2 VALUES('가나다','가나다');
7	INSERT INTO CHAR_TEST2 VALUES('123','123');
8	
9	SELECT * FROM CHAR_TEST2 WHERE COL1 = COL2;
10	

Result	
	Grid Result Server Output Text Output Explain Plan SQL Statistics
COL1	COL2
1	ABC
2	가나다

### CASE3) CHAR 타입과 문자상수 비교 시에도 위와 같음

SQL1 *	
1	SELECT COL1
2	FROM CHAR_TEST2
3	WHERE COL1 IN ('ABC', '가나다', '123');
4	

Result	
	Grid Result Server Output Text Output Explain Plan Statistics
COL1	
1	ABC
2	가나다

### ● VARCHAR 타입 컬럼의 문자 상수 비교

1. 왼쪽에서부터 서로 다른 문자가 나올 때 까지 비교하여 **공백포함 모든 값이 같을 때 같은 값으로 인정!**

### CASE) VARCHAR 타입끼리의 비교

SQL1 *	
1	INSERT INTO VARCHAR_TEST3 VALUES('ABC', 'ABCDE');
2	INSERT INTO VARCHAR_TEST3 VALUES('abc', 'abc');
3	INSERT INTO VARCHAR_TEST3 VALUES('DEFG', 'DEFG');
4	COMMIT;
5	
6	SELECT * FROM VARCHAR_TEST3 WHERE COL1 = COL2;
7	

Result	
	Grid Result Server Output Text Output Explain Plan Statistics
COL1	COL2
1	DEFG DEFG

CHAR 타입과는 다르게 공백까지 완전히 일치해야 같은 값으로 인정함(문자상수와 비교 시에도 동일함)

### ● 문자타입과 숫자 상수 비교

- 항상 숫자에 맞춰 형 변환 후 비교(매우 중요!!!!)

- 1) 문자 컬럼에 숫자변환이 불가능한 문자('ABCDE')가 이미 존재하는 경우 비교 불가
- 2) 문자 컬럼에 숫자변환이 가능한 문자('0001')만 존재하는 경우 비교 가능

### CASE) 문자 컬럼과 숫자 상수의 비교(이미 문자 컬럼에 숫자로 변경 불가능한 문자가 있는 경우)

< 테이블 데이터 >

SQL1 *	
1	SELECT * FROM CHAR_TEST1;
2	

Result	
	Grid Result Server Output Text Output Explain Plan Statistics
COL1	COL2
1	ABCD ABCD
2	abcd abcd
3	12345 12345
4	67890 67890

### < 값의 비교 >

SQL1 *	
1	SELECT * FROM CHAR_TEST1 WHERE col1 = '12345'; -- 조희가능
2	SELECT * FROM CHAR_TEST1 WHERE col1 = 12345; -- 조희불가
3	

Result	
Grid Result	Server Output
Text Output	Explain Plan
Statistics	

ORA-01722: 수치가 부적합합니다	
-----------------------	--

☞ 문자컬럼을 숫자상수와 비교 시 TO\_NUMBER 로 숫자로 변경하고 처리하는데 이미 숫자로 변환 불가능한 abcd 등의 데이터가 존재하므로 불가함

CASE) 문자 컬럼과 숫자 컬럼 비교(문자 컬럼에 숫자로 변경 가능한 문자만 있는 경우)

### < 테이블 데이터 >

SQL1 *	
1	SELECT * FROM CHAR_TEST2;
2	

Result	
Grid Result	Server Output
Text Output	Explain Plan
Statistics	

	COL1	COL2
1	12345	12345
2	67890	67890

### < 값의 비교 >

SQL1 *	
1	SELECT * FROM CHAR_TEST2 WHERE COL1 = '67890';
2	SELECT * FROM CHAR_TEST2 WHERE COL1 = 67890;
3	

Result	
Grid Result	Server Output
Text Output	Explain Plan
Statistics	

	COL1	COL2
1	67890	67890

● 숫자 컬럼과 문자 상수(숫자로 변경 가능한)와의 비교는 언제나 가능

- 문자와 숫자 비교 시 숫자에 맞춰 비교하게 되는데, 문자 상수를 숫자 타입으로 바꾸는 것은 제한이 없으므로 비교 가능!

CASE) 숫자 컬럼과 문자 상수의 비교

SQL1 \*

```

1 CREATE TABLE NUM_TEST1(
2   COL1 NUMBER
3 );
4
5 INSERT INTO NUM_TEST1 VALUES(123);
6 INSERT INTO NUM_TEST1 VALUES('0123');
7
8 SELECT * FROM NUM_TEST1 WHERE COL1 = '123';
  
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

	COL1
1	123
2	123

● 문자 컬럼에 숫자 상수 입력 가능, 숫자 컬럼에 숫자처럼 생긴 문자 상수 입력 가능

SQL1 \*

```

1 CREATE TABLE NUM_TEST2(
2   COL1 NUMBER,
3   COL2 VARCHAR(5)
4 );
5
6 INSERT INTO NUM_TEST2 VALUES('1234',1234);
7
8 SELECT * FROM NUM_TEST2;
  
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

	COL1	COL2
1	1234	1234



## EXISTS, NOT EXISTS

### ● EXISTS 연산자의 원리

- EXISTS 연산자는 IN 연산자처럼 포함 관계를 표현할 때 자주 등장
- EXISTS 연산자는 비교 컬럼을 따로 명시 않고 바로 서브쿼리에서 메인쿼리와 비교를 진행함

\*\* 문법

```
SQL1 *  
1 SELECT * FROM T1  
2 WHERE EXISTS (SELECT 'X' FROM T2  
3               WHERE T1.V1 = T2.V1);
```

- ☞ 메인쿼리 EXISTS 앞에 연산자 명시 불가
- ☞ 메인쿼리와 서브쿼리 비교 조건은 항상 서브쿼리에 명시(조건이 일치하는 T1의 값만 출력됨)
- ☞ 서브쿼리에서의 SELECT 절은 어떤 값도 입력 가능(자리를 채우기 위한 표현임)

※ 실제로 EXISTS는 조건절이 중요한 것이지 서브쿼리의 SELECT 절은 중요하지 않음

예제) 아래 두 테이블이 있을 때 T1의 V1 값이 T2의 V1 값과 일치하는 값 출력

SQL1 \*

1

2

3

SELECT \* FROM T1;

Result

Grid Result

Server Output

Text Output

	N1	V1
1	1	A
2	1	B
3	1	C

SQL1 \*

1

2

3

SELECT \* FROM T2;

Result

Grid Result

Server Output

Text Output

	N1	V1
1	1	A
2	3	B

결과)

```
SQL1 *  
1  SELECT * FROM T1  
2     WHERE EXISTS (SELECT 'X' FROM T2  
3                   WHERE T1.V1 = T2.V1);
```

Result

Grid Result   Server Output   Text Output   Explain Plan   Statistics

	N1	V1
1	1	A
2	1	B

- ☞ 이 때, 서브쿼리의 'X' 대신 어떤 값을 써도 결과는 동일  
T1.V1 = T2.V1을 확인하고 조건이 참이면 출력, 조건이 거짓이면 생략하는 기능에만 초점을 맞춘 문법이므로  
SELECT 결과는 중요하지 않음



따라서 아래와 같이 작성가능

SQL1 *	
1	SELECT * FROM T1
2	WHERE EXISTS (SELECT 1 FROM T2
3	WHERE T1.V1 = T2.V1);

Result		
	N1	V1
1	1	A
2	1	B

또한 IN 연산자로도 표현 가능하며 아래와 같이 작성 가능

SQL1 *	
1	SELECT * FROM T1
2	WHERE V1 IN (SELECT V1 FROM T2);

Result		
	N1	V1
1	1	A
2	1	B

IN 연산자는 반드시 비교할 컬럼(V1)을 명시해야 하며, 서브쿼리의 SELECT 절의 출력 결과와 비교하게 되므로 서브쿼리의 SELECT 절은 아무 값이나 전달하면 안됨!

## ● NOT EXISTS

- 메인쿼리의 결과를 출력하되, 서브쿼리 결과와 일치하지 않는 데이터만 출력할 목적으로 사용

만약 T2 테이블의 값을 제외하고자 한다면 아래와 같이 표현 가능

SQL1 *	
1	SELECT * FROM T1
2	WHERE NOT EXISTS (SELECT 'x'
3	FROM T2
4	WHERE T1.V1 = T2.V1);

Result		
	N1	V1
1	1	C

역시 NOT IN 연산자로도 표현 가능

SQL1 *	
1	SELECT * FROM T1
2	WHERE V1 NOT IN (SELECT V1 FROM T2);

Result		
	N1	V1
1	1	C

## ORACLE VS SQL Server(MSSQL)

\*\* ORACLE 과 SQL Server 는 구조, 기능적인 부분에서 차이가 발생, SQL 의 기본적인 표준은 비슷함.  
주로 다음의 차이가 발생(SQLD 기준 설명)

1. 조인 문법
2. 함수
3. 빈문자열 입력 처리
4. 문자열 입력 시 크기 제한
5. AUTO COMMIT
6. 테이블 별칭
7. TOP N QUERY

### ● 조인

- ORACLE 은 ORACLE 표준, ANSI 표준 모두 가능
- SQL Server 는 ANSI 표준만 가능

#### ORACLE VS SQL Server 조인 문법 비교

<pre>SQL1 * 1 SELECT * 2 FROM EMP, DEPT 3 WHERE EMP.DEPTNO = DEPT.DEPTNO 4 AND EMP.JOB = 'MANAGER';</pre>	일반 조인	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X SELECT * FROM EMP JOIN DEPT ON EMP.DEPTNO = DEPT.DEPTNO WHERE EMP.JOB = 'MANAGER';</pre>
<pre>SQL1 * 1 SELECT E1.ENAME AS 사원명, 2 E2.ENAME AS 상위관리자명 3 FROM EMP E1, EMP E2 4 WHERE E1.MGR = E2.EMPNO(+);</pre>	아우터 조인	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X SELECT * FROM EMP E1 LEFT OUTER JOIN EMP E2 ON E1.MGR = E2.EMPNO;</pre>

### ● 함수

- 두 DBMS 간의 함수가 가장 많은 차이 발생

#### ORACLE VS SQL Server 주요 함수 차이

<pre>SQL1 * 1 SELECT SAL, COMM, NVL(COMM,0) 2 FROM EMP;</pre>	널치환함수	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X SELECT SAL, COMM, ISNULL(COMM,0) FROM EMP;</pre>
<pre>SQL1 * 1 SELECT SUBSTR('ABCDE',2,2) AS 결과1, 2 INSTR('A#B#C#D', '#') AS 결과2 3 FROM DUAL;</pre>	문자열함수	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X SELECT SUBSTRING('ABCDE',2,2) AS 결과1, CHARINDEX('A#B#C#D', '#') AS 결과2</pre>
<pre>SQL1 * 1 SELECT CAST('2024/01/01' AS DATE), 2 TO_DATE('2024/01/01', 'YYYY/MM/DD') 3 FROM DUAL;</pre>	변환함수	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X SELECT CAST('2024/01/01' AS DATETIME), CONVERT(DATETIME, '2024/01/01')</pre>

## ● 빈문자열 입력 처리

- 빈문자열이 입력되면 **ORACLE** 은 **NULL** 로 입력, **SQL Server** 는 빈문자열 그대로 삽입됨

### ORACLE VS SQL Server 빈문자열 입력 차이

SQL1 *	빈문자열 입력	SQLQuery1.sql - DE...S.master (sa (53))*	빈문자열로 조회								
<pre>1 CREATE TABLE NULL_TEST1( 2   COL1 CHAR(5), 3   COL2 VARCHAR2(5) 4 ); 5 6 INSERT INTO NULL_TEST1 VALUES('','');</pre>	빈문자열 입력	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 CREATE TABLE NULL_TEST1( 2   COL1 CHAR(5), 3   COL2 VARCHAR(5) 4 ); 5 6 INSERT INTO NULL_TEST1 VALUES('','');</pre>									
<pre>1 SELECT * FROM NULL_TEST1 WHERE COL1 = '';</pre>	빈문자열로 조회	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 SELECT * FROM NULL_TEST1 WHERE COL1 = '';</pre>									
<table border="1"> <thead> <tr> <th>COL1</th> <th>COL2</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	COL1	COL2			조회 안됨	<table border="1"> <thead> <tr> <th>COL1</th> <th>COL2</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	COL1	COL2			조회 됨
COL1	COL2										
COL1	COL2										
<pre>1 SELECT * FROM NULL_TEST1 WHERE COL1 IS NULL;</pre>	NULL로 조회	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 SELECT * FROM NULL_TEST1 WHERE COL1 IS NULL;</pre>									
<table border="1"> <thead> <tr> <th>COL1</th> <th>COL2</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	COL1	COL2			NULL로 조회 시 출력됨	<table border="1"> <thead> <tr> <th>COL1</th> <th>COL2</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	COL1	COL2			NULL로 조회 시 출력 안됨
COL1	COL2										
COL1	COL2										
<pre>1 SELECT * FROM NULL_TEST1 WHERE COL2 IS NULL;</pre>	NULL로 조회	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 SELECT * FROM NULL_TEST1 WHERE COL2 IS NULL;</pre>									
<table border="1"> <thead> <tr> <th>COL1</th> <th>COL2</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	COL1	COL2			NULL로 조회	<table border="1"> <thead> <tr> <th>COL1</th> <th>COL2</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	COL1	COL2			NULL로 조회
COL1	COL2										
COL1	COL2										

## ● 문자열 입력 시 크기 제한

- **ORACLE** 은 정해진 사이즈를 초과한 문자열 입력 불가
- **SQL Server** 는 공백을 포함한 문자열 입력 시 사이즈 초과만큼의 공백을 자르고 입력  
(SQL Server : SET ANSI\_PADDING 파라미터 설정에 따라 달라질 수 있음)

### ORACLE VS SQL Server 문자열 입력 시 크기 제한 비교

SQL1 *	제한 크기를 초과하는 공백을 포함한 문자열 삽입 처리	SQLQuery1.sql - DE...S.master (sa (53))*	정상
<pre>1 CREATE TABLE INSERT_TEST1( 2   COL1 CHAR(5), 3   COL2 VARCHAR2(5) 4 ); 5 6 INSERT INTO INSERT_TEST1(COL1) VALUES('12345 ');</pre>	제한 크기를 초과하는 공백을 포함한 문자열 삽입 처리	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 CREATE TABLE INSERT_TEST1( 2   COL1 CHAR(5), 3   COL2 VARCHAR(5) 4 ); 5 6 INSERT INTO INSERT_TEST1(COL1) VALUES('12345 ');</pre>	
<pre>1 INSERT INTO INSERT_TEST1(COL2) VALUES('12345 ');</pre>	ORA-12899: "SCOTT"."INSERT_TEST1"."COL1" 열에 대한 값이 너무 큼(실제: 7, 최대값: 5) 여러 발생	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 INSERT INTO INSERT_TEST1(COL2) VALUES('12345 ');</pre>	(1개 행이 영향을 받음)
<pre>1 INSERT INTO INSERT_TEST1(COL2) VALUES('12345 ');</pre>	ORA-12899: "SCOTT"."INSERT_TEST1"."COL2" 열에 대한 값이 너무 큼(실제: 7, 최대값: 5)	<pre>SQLQuery1.sql - DE...S.master (sa (53))* X 1 INSERT INTO INSERT_TEST1(COL2) VALUES('12345 ');</pre>	(1개 행이 영향을 받음)

## ● AUTO COMMIT

- ORACLE 은 DDL 만 AUTO COMMIT (DML 의 경우 수동 COMMIT)
- SQL Server 는 DML, DDL 모두 AUTO COMMIT 이 기본
  - > TRANSACTION 선언으로 TRANSACTION 별 수동 COMMIT/ROLLBACK 처리 가능
- 세션별/IDE 별 설정 변경 가능

### ORACLE VS SQL Server AUTO COMMIT 비교

SQL1 \*

```

1 CREATE TABLE TRANSACTION1(
2   COL1 NUMBER
3 );
4
5 INSERT INTO TRANSACTION1 VALUES(1);
6 INSERT INTO TRANSACTION1 VALUES(2);
7
8 ALTER TABLE TRANSACTION1 ADD COL2 CHAR(5);
9 ROLLBACK;
10
11 SELECT * FROM TRANSACTION1;
  
```

데이터 입력

DDL 시 AUTO COMMIT

Result

	COL1	COL2
1	1	
2	2	

SQL1 \*

```

1 CREATE TABLE TRANSACTION2(
2   COL1 NUMBER
3 );
4
5 INSERT INTO TRANSACTION2 VALUES(1);
6 INSERT INTO TRANSACTION2 VALUES(2);
7
8 ALTER TABLE TRANSACTION2 ADD COL2 CHAR(5);
9
10 INSERT INTO TRANSACTION2(COL1) VALUES(3);
11 INSERT INTO TRANSACTION2(COL1) VALUES(4);
12
13 ROLLBACK;
14
15 SELECT * FROM TRANSACTION2;
  
```

이 두 건의 INSERT만 ROLLBACK 처리 됨

Result

	COL1	COL2
1	1	
2	2	

SQLQuery1.sql - DE...S.master (sa (53))\* × 세션1 수행

```

1 CREATE TABLE TRANSACTION1(
2   COL1 NUMERIC
3 );
4
5 INSERT INTO TRANSACTION1 VALUES(1);
6 INSERT INTO TRANSACTION1 VALUES(2);
  
```

100 %

메시지

(1개 행이 영향을 받음)

(1개 행이 영향을 받음)

SQLQuery1.sql - DE...S.master (sa (56))\* × 세션2 수행

```

1 SELECT * FROM TRANSACTION1;
  
```

100 %

결과 메시지

	COL1
1	1
2	2

DML AUTO COMMIT 됨

SQLQuery1.sql - DE...S.master (sa (56))\* × SQL Server TRANSACTION 제어 방법

```

1 BEGIN TRAN
2 INSERT INTO TRANSACTION1 VALUES(3);
3 INSERT INTO TRANSACTION1 VALUES(4);
4 ROLLBACK;
5 SELECT * FROM TRANSACTION1;
  
```

100 %

결과 메시지

	COL1
1	1
2	2

3, 4번 값 ROLLBACK 처리 됨

## ● 테이블 별칭

- **ORACLE** 은 **AS 없이 전달**(AS 전달 시 에러)
- **SQL Server** 는 테이블 별칭 시 **AS 를 전달 혹은 생략 가능** 하지만, 인라인뷰에는 반드시 AS 를 전달해야 함

### ORACLE VS SQL Server 테이블 별칭 전달 차이

```
SQL1 *
1 SELECT ENAME, SAL, DEPTNO
2 FROM (SELECT * FROM EMP) E;
3
```

AS 생략 시

```
SQL1 *
1 SELECT ENAME, SAL, DEPTNO
2 FROM (SELECT * FROM EMP) AS E;
3
```

AS 전달 시

Result

Grid Result Server Output Text Output Explain Plan SQL

ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다

```
SQL1 *
1 SELECT ENAME, SAL, DEPTNO
2 FROM (SELECT * FROM EMP);
3
```

ALIAS 생략 시

```
SQLQuery1.sql - DE...S.master (sa (53))* X
SELECT ENAME, SAL, DEPTNO
FROM (SELECT * FROM EMP) E;
```

```
SQLQuery1.sql - DE...S.master (sa (53))* X
SELECT ENAME, SAL, DEPTNO
FROM (SELECT * FROM EMP) AS E;
```

SQLQuery1.sql - DE...S.master (sa (53))\* X

```
SELECT ENAME, SAL, DEPTNO
FROM (SELECT * FROM EMP);
```

100 %

메시지

메시지 102, 수준 15, 상태 1, 줄 2  
';' 근처의 구문이 잘못되었습니다.

## ● TOP N QUERY

- **ORACLE** 은 **ROWNUM** 방식
- **SQL Server** 는 TOP 쿼리 방식(**TOP n [WITH TIES]**)
- 둘 다 RANK 함수와 FETCH 절 사용 가능(OFFSET 도 가능)

### ORACLE VS SQL Server TOP N QUERY

```
SQL1 *
1 SELECT ENAME, SAL, DEPTNO
2 FROM (SELECT *
3 FROM EMP
4 ORDER BY SAL DESC)
5 WHERE ROWNUM <= 2;
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

	ENAME	SAL	DEPTNO
1	KING	5000	10
2	SCOTT	3000	20

동순위를 출력하기 위해서는 RANK 사용

```
SQLQuery1.sql - DE...S.master (sa (53))* X
SELECT TOP 2 ENAME, SAL, DEPTNO
FROM EMP
ORDER BY SAL DESC;
```

100 %

결과 메시지

	ENAME	SAL	DEPTNO
1	KING	5000.00	10
2	SCOTT	3000.00	20

```
SQLQuery1.sql - DE...S.master (sa (53))* X
SELECT TOP 2 WITH TIES ENAME, SAL, DEPTNO
FROM EMP
ORDER BY SAL DESC;
```

100 %

결과 메시지

	ENAME	SAL	DEPTNO
1	KING	5000.00	10
2	FORD	3000.00	20
3	SCOTT	3000.00	20

동 순위 모두 출력

RANK OVER 함수와 FETCH 절 사용법은 같다.



## ● DDL

- 컬럼 수정, DEFAULT 값 변경, 컬럼 이름 변경의 문법에서 차이 발생(컬럼 추가 및 삭제는 동일함)

### ORACLE VS SQL Server DDL 문법 차이

SQL1 *	1 CREATE TABLE DDL_TEST( 2 COL1 NUMBER NOT NULL 3 ); 4 5 ALTER TABLE DDL_TEST ADD COL2 VARCHAR2(50); 6	컬럼 추가
SQL1 *	1 ALTER TABLE DDL_TEST DROP COLUMN COL2; 2 3	컬럼 삭제
SQL1 *	1 ALTER TABLE DDL_TEST MODIFY COL2 NUMBER(10); 2 3	컬럼 수정 (타입, 크기)
SQL1 *	1 ALTER TABLE DDL_TEST RENAME COLUMN COL1 TO COL11; 2 3	컬럼 이름 변경
SQL1 *	1 ALTER TABLE DDL_TEST MODIFY COL11 DEFAULT 1000; 2 3	DEFAULT값 수정

SQLQuery1.sql - DE...S.master (sa (52))* X	1 CREATE TABLE DDL_TEST( 2 COL1 NUMERIC NOT NULL 3 ); 4 5 ALTER TABLE DDL_TEST ADD COL2 VARCHAR(50);	
SQLQuery1.sql - DE...S.master (sa (52))* X	1 ALTER TABLE DDL_TEST DROP COLUMN COL2; 2	
SQLQuery1.sql - DE...S.master (sa (52))* X	1 ALTER TABLE DDL_TEST ALTER COLUMN COL2 NUMERIC(10); 2	
SQLQuery1.sql - DE...S.master (sa (52))* X	1 EXEC SP_RENAME 'DDL_TEST.COL1', 'COL11', 'COLUMN'; 2	
SQLQuery1.sql - DE...S.master (sa (52))* X	1 ALTER TABLE DDL_TEST ADD CONSTRAINT DEFAULT_CONS 2 DEFAULT 1000 FOR COL11;	

## ● 컬럼 수정 시 NULL 설정 여부

- 컬럼 생성 시 DEFAULT 는 Nullable
- ORACLE 은 NOT NULL 속성 컬럼 타입이나 크기를 변경해도 NOT NULL 속성을 유지, SQL Server 는 Nullable 로 변경됨(다시 NOT NULL 선언 필요)






### ORACLE VS SQL Server NULL 속성 처리

```

SQL1 *
1 CREATE TABLE DDL_TEST2(
2   COL1 CHAR(10) NOT NULL
3 );
4
5 ALTER TABLE DDL_TEST2 MODIFY COL1 CHAR(5);
6

```

Result

 Grid Result
  Server Output
  Text Output
  Explain Plan
  Statistics

Column	Nullable	Type	Comment
1 COL1	NOT NULL	CHAR(5)	

Table DDL\_TEST2 described.

SQLQuery1.sql - DE...S.master (sa (56))* X	1 CREATE TABLE DDL_TEST2( 2 COL1 CHAR(10) NOT NULL 3 ); 4 5 ALTER TABLE DDL_TEST2 ALTER COLUMN COL1 CHAR(5); 6 EXEC sp_help 'DDL_TEST2';
100 %	결과 메시지
1	Name Owner Type Created_datetime
1	DDL_TEST2 dbo user table 2024-02-29 19:23:263
1	Column_name Type Computed Length Prec Scale Nullable TrimTrailingBlanks FixedL
1	COL1 char no 5 5 5 yes no yes
SQLQuery1.sql - DE...S.master (sa (52))* X	1 ALTER TABLE DDL_TEST2 ALTER COLUMN COL1 CHAR(5) NOT NULL; 2 EXEC sp_help 'DDL_TEST2';
100 %	결과 메시지
1	Name Owner Type Created_datetime
1	DDL_TEST2 dbo user table 2024-02-29 19:13:45,310
1	Column_name Type Computed Length Prec Scale Nullable TrimTrailingBlanks FixedL
1	COL1 char no 5 5 5 no no no