목차

- 2.8.1. 질의 변형
- 2.8.2. 최적화 방법
- 2.8.3. 접근 방법
- 2.8.4. 조인 순서
- 2.8.5. 조인 방법
- 2.8.6. 병렬 처리
- 2.8.7. 실체화 뷰

2.9. 스키마 객체

- 2.9.1. 테이블
- 2.9.2 인덱스
- 2.9.3. 뷰
- 2.9.4. 시퀀스
- 2.9.5. 동의어

2.1. 데이터 타입

- 내용
  - 종류
  - 특징
  - 시나리오 수행
- 종류

데이터 타입	종류
문자형	CHAR, VARCHAR, VARCHAR2, NCHAR, NVARCHAR, NVARCHAR2, RAW, LONG, LONG RAW
숫자형	NUMBER, INTEGER, FLOAT, BINARY_FLOAT, BINARY_DOUBLE
날짜형	DATE, TIME, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE
간격형	INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
대용량 객체형	CLOB, BLOB, XMLTYPE
내재형	ROWID
사용자 정의형	배열, 네스티드 테이블

2.1.1.문자형

CHAR

- `CHAR(size[BYTE|CHAR])`
- 문자열을 저장하는 데이터 타입이고 항상 고정된 문자열 길이를 갖는다.
- 문자열의 길이는 byte 와 문자를 기준으로 지정할 수 있다.
- 문자열은 최대 2,000byte 나 2,000 자까지 선언할 수 있다.

VARCHAR

- `VARCHAR(size[BYTE|CHAR])`
- 문자열을 저장하는 데이터 타입이고 문자열 길이가 일정하지 않은 가변 길이를 갖는다.
- 문자열의 길이는 byte 와 문자를 기준으로 지정할 수 있다.
- 선언된 문자열 길이의 범위 내에서 입력된 문자열 길이와 동일한 길이를 갖는다.
- 문자열은 최대 65,532byte 나 65,532 자까지 선언할 수 있다.

VARCHAR2

- **VARCHAR** 과 완전히 동일하다.

시나리오 수행_고정형, 가변형 차이 보기

시나리오 내용
테이블 생성
데이터 입력
DECODE 함수는 IF 문 같은 함수 DECODE(컬럼, '비교값', '같으면', '다르면')

시나리오 수행내역
테이블 생성 후 데이터 입력

```
CREATE TABLE TEST_CHARACTER(
 T_CHAR CHAR(4),
 T_VARCHAR VARCHAR(4),
 T_VARCHAR2 VARCHAR2(4)
);

INSERT INTO TEST_CHARACTER VALUES('A', 'A', 'A');
```

```
SQL> CREATE TABLE TEST_CHARACTER(
 T_CHAR CHAR(4),
 T_VARCHAR VARCHAR(4),
 T_VARCHAR2 VARCHAR2(4)
);   2    3    4    5

Table 'TEST_CHARACTER' created.

SQL> INSERT INTO TEST_CHARACTER VALUES('A', 'A', 'A');

1 row inserted.
```

```
SELECT  DECODE(T_CHAR, 'A   ', 'TRUE', 'FALSE')          T_CHAR,
        DECODE(T_VARCHAR, 'A   ', 'TRUE', 'FALSE')     T_VARCHAR,
        DECODE(T_VARCHAR2, 'A   ', 'TRUE', 'FALSE')   T_VARCHAR2
FROM   TEST_CHARACTER;
```

```
SQL> SELECT  DECODE(T_CHAR, 'A   ', 'TRUE', 'FALSE')          T_CHAR,
           DECODE(T_VARCHAR, 'A   ', 'TRUE', 'FALSE')     T_VARCHAR,
           DECODE(T_VARCHAR2, 'A   ', 'TRUE', 'FALSE')   T_VARCHAR2
FROM   TEST_CHARACTER;   2    3    4

T_CHAR T_VARCHAR T_VARCHAR2
------ --------- ----------
TRUE   FALSE     FALSE

1 row selected.
```

NCHAR

- `NCHAR(size)`
- 유니코드 문자열을 저장하기 위한 타입이고 항상 고정된 문자열 길이를 갖는다.
- 기본적으로 **CHAR** 타입과 유사하지만, 문자열의 길이가 문자 기준이다.
- 문자열의 최대 길이는 2,000 자이다.

- 문자열의 길이가 0 인 값은 NULL 로 인식된다.

NVARCHAR

- `NVARCHAR(size)`
- 기본적으로 **VARCHAR** 타입과 유사하지만, 문자열의 길이가 문자 기준이다.
- 문자열의 최대 길이는 65,532 자이다. 단, 65,532byte 를 초과할 수 없다.
- 문자열의 길이가 0 인 값은 NULL 로 인식된다.

NVARCHAR2

- **NVARCHAR** 과 완전히 동일하다.

RAW

- `RAW(size)`
- 임의의 바이너리 데이터를 저장하는 데이터 타입이다.
- 최대 2,000byte 까지 선언할 수 있다.
- 선언된 길이 내에서 가변 길이를 갖는다.
- 데이터 중간에 NULL 문자('\0')가 올 수 있다.
- 입출력을 수행할 때 **RAW** 타입의 데이터는 16 진수로 표현된다.
  ex] 4byte 의 데이터는 16 진수로 '012345AB'로 표현되며 필요한 경우 맨 앞이 0 으로 시작되어야 한다.

LONG

- `LONG`
- **VARCHAR** 타입을 확장한 데이터 타입이다. (일반 문자열 저장)
- 최대 2GB 까지 선언할 수 있다.
- 테이블 내의 한 컬럼에만 선언할 수 있다.
- 컬럼에 대해서는 인덱스를 생성할 수 없다.
- **LONG** 타입의 컬럼을 포함한 로우(Row)가 디스크에 저장될 때에는 다른 컬럼의 값과 함께 동일한 디스크 블록에 저장되며, 길이에 따라 여러 디스크 블록에 걸쳐 저장될 수 있다.
- **LONG** 타입의 데이터에 접근할 때는 항상 순차적으로만 접근할 수 있으며, 임의의 위치에 대해 연산은 할 수 없다.

LONG RAW

- `LONG RAW`
- RAW 타입을 확장한 데이터 타입이다. (임의의 바이너리 데이터 저장)

- 최대 2GB 까지 선언할 수 있다.
- 테이블 내의 한 컬럼에만 선언할 수 있다.
- 컬럼에 대해서는 인덱스를 생성할 수 없다.
- **LONG RAW** 타입의 컬럼을 포함한 로우가 디스크에 저장될 때에는 다른 컬럼의 값과 함께 동일한 디스크 블록에 저장되며, 길이에 따라 여러 디스크 블록에 걸쳐 저장될 수 있다.
- **LONG RAW** 타입의 데이터에 접근할 때는 항상 순차적으로만 접근할 수 있으며, 임의의 위치에 대해 연산은 할 수 없다.

→ LONG, LONG RAW 는 UNIQUE, PRIMARY KEY, FOREIGN KEY 제약조건의 키 컬럼에 포함될 수 없다.

시나리오 수행_RAW,Long

시나리오 내용
RAW 테이블
Index 삽입
RAW 테이블 - ERROR
LONG 테이블
INDEX 생성

시나리오 수행내역
RAW 테이블

```
CREATE TABLE RAWTAB(
COL1 RAW(2000));

INSERT INTO RAWTAB VALUES(HEXTORAW('2ADD'));
INSERT INTO RAWTAB VALUES('2A7F');
INSERT INTO RAWTAB VALUES('0010101001111111');

SELECT * FROM RAWTAB;
```

```
SQL> CREATE TABLE RAWTAB(
  2 COL1 RAW(2000));

Table 'RAWTAB' created.

SQL> INSERT INTO RAWTAB VALUES(HEXTORAW('2ADD'));

1 row inserted.

SQL> INSERT INTO RAWTAB VALUES('2A7F');

1 row inserted.

SQL> INSERT INTO RAWTAB VALUES('0010101001111111');

1 row inserted.

SQL> SELECT * FROM RAWTAB;

COL1
--------------------
2ADD
2A7F
0010101001111111

3 rows selected.
```

RAW 테이블 최대길이 초과

```
CREATE TABLE RAWTAB_ERR(
COL1 RAW(4000));
```

```
SQL> CREATE TABLE RAWTAB_ERR(
   2 COL1 RAW(4000));
TBR-5079: Data type length is out of range.
at line 2, column 10 of null:
COL1 RAW(4000))
         ^^^^
```

t 테이블 생성 후 index 삽입

```
create table t (a number, b long);
create unique index i in t(a, b);
```

```
SQL> CREATE TABLE t (a NUMBER, b LONG);

Table 'T' created.

SQL> ^C
SQL> CREATE UNIQUE INDEX i ON t(a, b);
TBR-8063: LONG columns are not permitted.
at line 1, column 33 of null:
CREATE UNIQUE INDEX i ON t(a, b)
                                ^
```

2.1.2.숫자형

NUMBER

- `NUMBER[(precision[,scale])]`
- 정수 또는 실수를 저장하는 데이터 타입이다.
- 정밀도(precision)와 스케일(scale)을 생략하여 선언한 경우 표현 가능한 최대 범위와 최대 정밀도 내에서 임의의 자릿수를 갖는 모든 데이터 값을 지원한다.
    - precision
        - 정밀도는 유효숫자의 최대 자릿수이다.
        - 정밀도는 1 ~ 38 까지 정의할 수 있다.

- scale
  - 스케일은 소수점 아래 가장 오른쪽 유효숫자까지의 자릿수이다.
  - 스케일은 -125 ~ 130 까지 정의할 수 있다.

시나리오 수행_NUMBER

시나리오 내용
테스트 테이블 생성
데이터 입력

시나리오 수행내역
num1 테이블 생성
create table num1 (a number); insert into num1 values(12345.678); select * from num1; SQL> create table num1 (a number); Table 'NUM1' created. SQL> insert into num1 values(12345.678); 1 row inserted. SQL> select * from num1; A ---------- 12345.678 1 row selected.
num2 테이블 생성

```
/* 정밀도에 *(아스트릭스)를 붙여서 시스템한테 알아서하도록 의뢰 */
create table num2 (a number(*,3));
insert into num2 values(12345.678);
select * from num2;
```

```
SQL> create table num2 (a number(*,3));

Table 'NUM2' created.

SQL> insert into num2 values(12345.678);

1 row inserted.

SQL> select * from num2;

         A
----------
 12345.678

1 row selected.
```

num3 테이블 생성

```
create table num3 (a number(8,-2));
insert into num3 values(12345.678);
select * from num3;
```

```
SQL> create table num3 (a number(8,-2))
  2 ;

Table 'NUM3' created.

SQL> insert into num3 values(12345.678);

1 row inserted.

SQL> select * from num3;

         A
----------
     12300

1 row selected.
```

num4 테이블 생성

```
create table num4 (a number(3));
insert into num4 values(12345.678);
select * from num4;
/* 오류 정수부분 자릿수 부족 */
```

```
SQL> create table num4 (a number(3));

Table 'NUM4' created.

SQL> insert into num4 values(12345.678);
TBR-5111: NUMBER exceeds given precision. (n:12346, p:3, s:0)
```

BINARY_FLOAT 타입

- 실수나 정수를 표현하고, 32 비트로 저장하는 단일 정밀도 데이터 타입이다.
- 특별값인 INF, -INF, NaN(Not A Number)을 지원한다.
- 사칙연산을 모두 지원한다.
- 비교 연산자를 지원한다. 단, NaN 은 다른 모든 값보다 가장 큰 값으로 취급하고, 서로 다른 NaN 과 NaN 은 같다.
- 각종 변환함수 및 수학함수를 지원한다.

BINARY_DOUBLE 타입

- 실수나 정수를 표현하고, 64 비트로 저장하는 2 배 정밀도 데이터 타입이다.

→ 숫자형 타입의 우선순위 : BINARY_DOUBLE > BINARY_FLOAT > NUMBER

2.1.3.날짜형

DATE

- DATE
- 연도, 월, 일, 시, 분, 초를 표현할 수 있다.
- 연도는 BC 9,999 ~ AD 9,999 까지 표현할 수 있다.
- 시간은 24 시간 단위로 표현된다.

TIME

- TIME [(fractional_seconds_precision)]
- 초 단위 소수점 9 자리까지의 특정 시간을 표현하는 데이터 타입이다.
- fractional_seconds_precision : 초 단위의 소수점 자릿수이다. 0~9 사이의 값을 사용할 수 있다. (기본값 : 6)

TIMESTAMP

- `TIMESTAMP [(fractional_seconds_precision)]`
- 기본 날짜형을 확장한 자료형 (시간대 정보없는 데이터 타입)
- 날짜와 초 단위 소수점 9 자리까지의 시간을 모두 표현하는 데이터 타입이다.

TIMESTAMP WITH TIME ZONE

- `TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE`
- 서버가 위치한 시간대 정보포함
- TIMESTAMP WITH TIME ZONE 타입은 TIMESTAMP 타입을 확장하여 시간대까지 표현하는 데이터 타입이다.

TIMESTAMP WITH LOCAL TIME ZONE

- `TIMESTAMP [(fractional_seconds_precision)] WITH LOCAL TIME ZONE`
- 클라이언트가 위치한 시간대 정보포함
- 특정 세션의 시간대에 따라 다르게 시간정보를 표현하는 데이터 타입이다.

시나리오 수행_DATE, TIMESTAMP

시나리오 내용
DATE 테이블 생성
데이터 입력
TIMESTAMP 테이블 생성
데이터 입력

시나리오 수행내역
DTAB 테이블 생성 후 데이터 입력

```
CREATE TABLE DTAB(
COL1 DATE,
COL2 DATE,
COL3 DATE
);

INSERT INTO DTAB
VALUES(SYSDATE, SYSDATE-30, SYSDATE+30);

INSERT INTO DTAB
VALUES('20191019', '20110910', '20310930');
```

```
SQL> CREATE TABLE DTAB(
COL1 DATE,
COL2 DATE,
COL3 DATE
);   2    3    4    5

Table 'DTAB' created.

SQL> INSERT INTO DTAB
VALUES(SYSDATE, SYSDATE-30, SYSDATE+30);    2

1 row inserted.

SQL> INSERT INTO DTAB
VALUES('20191019', '20110910', '20310930');

1 row inserted.
```

DTAB 테이블 조회

```
COL COL1 FOR A20
COL COL2 FOR A20
COL COL3 FOR A20
SET LINESIZE 100
SELECT * FROM DTAB;
```

```
SQL> COL COL1 FOR A20
SQL> COL COL2 FOR A20
SQL> COL COL3 FOR A20
SQL> SET LINESIZE 100
SQL> SELECT * FROM DTAB;

COL1                 COL2                 COL3
-------------------- -------------------- --------------------
2022/11/08           2022/10/09           2022/12/08
2019/10/19           2011/09/10           2031/09/30

2 rows selected.
```

TIMESTAMP 테이블 생성 및 데이터 입력

```
CREATE TABLE DTAB2(
COL1 TIMESTAMP,
COL2 TIMESTAMP WITH TIME ZONE,
COL3 TIMESTAMP WITH LOCAL TIME ZONE);

INSERT INTO DTAB2
VALUES(SYSDATE,SYSDATE,SYSDATE);
```

```
SQL> CREATE TABLE DTAB2(
  2 COL1 TIMESTAMP,
  3 COL2 TIMESTAMP WITH TIME ZONE,
  4 COL3 TIMESTAMP WITH LOCAL TIME ZONE);

Table 'DTAB2' created.

SQL> INSERT INTO DTAB2
  2 VALUES(SYSDATE,SYSDATE,SYSDATE);

1 row inserted.
```

TIMESTAMP 테이블 조회

```
SELECT * FROM DTAB2;
```

```
COL1                    COL2                    COL3
----------------------- ----------------------- -----------------------
2022/11/08 15:18:23.    2022/11/08 15:18:23.    2022/11/08 15:18:23.
000000                  000000 Asia/Seoul       000000
```

2.1.4. 대용량 객체형

- 대용량의 객체를 저장하기 위해 Tibero 에서 제공하는 가장 큰 데이터 타입이며, CLOB 타입과 BLOB 타입, XMLTYPE 타입이 있다.

BLOB

- LONG 타입을 확장한 데이터 타입이다. (이진데이터 처리)
- 데이터를 최대 4GB 까지 저장할 수 있다.
- 원본자료(사진, 동영상 등)를 데이터베이스 내부에 저장한다.

Q. BLOB 타입의 칼럼이 있는 table 을 생성해서 프로시저로 이미지를 가져오고 싶었는데 디렉토리를 못찾는다고 하는데 어떻게 설정해야하나요?

- 저장순서

1. 테이블 준비

2. 디렉토리 객체 생성

3. 데이터 삽입(INSERT 문으로 직접 삽입할 수 없다 때문에 여러개의 변수 필요

절차적 언어 PL_SQL 써야함, 익명블록(PL/SQL 문) 등 사용)

```
SQL 문
```

```
CREATE TABLE TEMP10(COL BLOB);

/*procedure*/

DECLARE

L_DIR VARCHAR2(20):= 'TEST_DIR';

L_FILE VARCHAR2(30):= 'sample.jpg';

L_BFILE BFILE;

L_BLOB BLOB;



BEGIN

INSERT INTO TEMP10(COL) VALUES(EMPTY_BLOB())

RETURN COL INTO L_BLOB;



-- A INTO B 는 B:=A A 값을 B 에 할당해라



L_BFILE:=BFILENAME(L_DIR,L_FILE);

DBMS_LOB.FILEOPEN(L_BFILE, DBMS_LOB.FILE_READONLY);

-- FILEOPEN -> B_FILE 에 있는 데이터를 READONLY 형식으로 열어라.
읽기전용이아니면 그림파일이 수정될 수 있음

DBMS_LOB.LOADFROMFILE(L_BLOB, L_BFILE,
DBMS_LOB.GETLENGTH(L_BFILE));

-- B_FILE 이 가지고 있는 길이를 정해서 B_FILE 에 있는 데이터를 꺼내서
BLOB=데이터베이스 안에 저장하라.

DBMS_LOB.FILECLOSE(L_BFILE);



COMMIT;

END;
/
```

ERROR 문

```
TBR-14053: Specified directory was not found.
TBR-15163: Unhandled exception at  line 19.
```

2.2. 데이터 타입 변환

2.2.1. 명시적 타입 변환

- 사용자가 SQL 변환 함수를 직접 사용하여 타입을 변환할 수 있다.
- 종류 : TO_CHAR, TO_NUMBER, TO_DATE

2.2.2. 암시적 타입 변환

- 사용자가 명시적으로 타입을 변환하지 않더라도, 필요하다면 암시적으로 타입을 변환하여 준다.
- 암시적 타입 변환이 필요한 경우는 아래와 같다.
    - 컬럼에 다른 타입의 데이터를 INSERT, UPDATE 하는 경우
    - 조건문에서 비교하는 양쪽 값이 다른 타입인 경우

2.3. 리터럴

- 리터럴은 상수 값을 나타내는 단어로 SQL 문에서 연산식이나 조건식의 일부로 사용된다. 사용될 문자 또는 날짜에 작은 따옴표(')로 표기한다.
- 종류 : 문자열, 숫자열, 간격, 날짜형 리터럴

2.4. 형식 문자열

2.4.1. NUMBER 타입

함수	설명
TO_CHAR	NUMBER 타입의 값을 문자열로 변환한다.
TO_NUMBER	문자열을 NUMBER 타입의 값으로 변환한다.

- 여러 가지 형식 요소로 구성된다. 소수점 위아래의 자릿수, 음양 부호의 출력, 쉼표(,) 또는 지수 형식 등을 출력할 수 있다.
- 화폐 단위를 나타내는 기호($, W 등)를 삽입할 수 있다.
- 16 진수로 출력할 수 있다.

- 별도의 문자열을 삽입할 수 없다.
- 대소문자를 구분하는 형식 요소가 없다.

2.4.2. 날짜형 타입

함수	설명
TO_CHAR	날짜형 타입의 값을 문자열로 변환한다.
TO_DATE	문자열을 날짜형 타입의 값으로 변환한다.
TO_TIMESTAMP	문자열을 날짜/시간형 타입의 값으로 변환한다.
TO_TIMESTAMP_TZ	문자열을 시간대를 포함하는 날짜/시간형 타입의 값으로 변환한다.

2.4.3. 형식 조절자

- FM 형식 조절자를 통해 공백을 채우는 방식을 변경할 수 있고, FX 형식 조절자를 통해 입력 문자열과 형식 문자열이 정확히 일치하는지 검사할 수 있다.

FM

Tibero 는 각각의 형식 요소에 대해 그 형식 요소가 출력하는 문자열의 최대 크기만큼 공백 문자를 채운다. 예를 들어 MONTH 형식 요소의 경우, 가장 긴 달은 'SEPTEMBER'이므로 나머지 달은 오른쪽에 공백을 채워서 아홉 글자를 맞추게 된다.

FX

Tibero 는 형식 문자열과 입력 문자열이 정확히 일치하는지 검사하고, 만약 하나라도 어긋나는 경우엔 에러를 발생시킨다.

2.5. 의사 칼럼

- 사용자가 명시적으로 선언하지 않아도, Tibero 시스템이 자동으로 모든 테이블에 포함하는 컬럼이다.

2.5.1. ROWID

- 전체 데이터베이스 내의 하나의 로우를 유일하게 참조하는 식별자이다.
- ROWID 는 전체 12byte 로 구성되어 있으며, Segment, Data File, Data Block, Row 가 각각 4, 2, 4, 2byte 로 되어 있다.

2.5.2. ROWNUM

- ROWNUM 은 SELECT 문장의 실행 결과로 나타나는 로우에 대하여 순서대로 번호를 부여한다.

시나리오 수행_ROWID, ROWNUM

시나리오 내용
TEST 사용자 생성
DEPARTMENT 테이블 생성 및 데이터 입력
EMPLOYEE 테이블 생성 및 데이터 입력
테이블 조회
ROWID 출력
ROWNUM 출력

시나리오 수행내역
TEST 사용자 생성
CREATE USER TEST IDENTIFIED BY TEST; GRANT CONNECT, GRANT TO TEST; CONN TEST/TEST SQL> CREATE USER TEST IDENTIFIED BY TEST; User 'TEST' created. SQL> GRANT CONNECT, RESOURCE TO TEST; Granted. SQL> CONN TEST/TEST Connected to Tibero.
DEPARTMENT 테이블 생성 및 데이터 입력

```
/*DEPARTMENT TABLE*/
CREATE TABLE DEPARTMENT
(DEPTNO NUMBER(2) CONSTRAINT DEP_DEPTNO_NN NOT NULL,
DNAME VARCHAR2(14), LOC VARCHAR2(13),
CONSTRAINT DEP_ID_PK PRIMARY KEY(DEPTNO));

INSERT INTO DEPARTMENT VALUES(10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPARTMENT VALUES(20,'RESEARCH','DALLAS');
INSERT INTO DEPARTMENT VALUES(30,'SALES','CHICAGO');
INSERT INTO DEPARTMENT VALUES(40,'OPERATIONS','BOSTON');
```

```
SQL> CREATE TABLE DEPARTMENT
(DEPTNO NUMBER(2) CONSTRAINT DEP_DEPTNO_NN NOT NULL,
DNAME VARCHAR2(14),
LOC VARCHAR2(13),
CONSTRAINT DEP_ID_PK PRIMARY KEY(DEPTNO));   2    3    4    5

Table 'DEPARTMENT' created.

SQL> INSERT INTO DEPARTENT VALUES(10,'ACCOUNTING','NEW YORK');
TBR-8033: Specified schema object was not found.
at line 1, column 14 of null:
INSERT INTO DEPARTENT VALUES(10,'ACCOUNTING','NEW YORK')
            ^

SQL> INSERT INTO DEPARTMENT VALUES(10,'ACCOUNTING','NEW YORK');

1 row inserted.

SQL> INSERT INTO DEPARTMENT VALUES(20,'RESEARCH','DALLAS');

1 row inserted.

SQL> INSERT INTO DEPARTMENT VALUES(30,'SALES','CHICAGO');

1 row inserted.

SQL> INSERT INTO DEPARTMENT VALUES(40,'OPERATIONS','BOSTON');
```

EMPLOYEE 테이블 생성 및 데이터 입력

```
/*EMPLOYEE TABLE*/
CREATE TABLE EMPLOYEE
(EMPNO NUMBER(4) CONSTRAINT EMP_EMPNO_NN NOT NULL,
ENAME VARCHAR2(15) CONSTRAINT EMP_ENAME_NN NOT NULL,
JOB VARCHAR2(15),
MANAGERNO NUMBER(4),
STARTDATE DATE,
SALARY NUMBER(10),
DEPTNO NUMBER(3),
CONSTRAINT EMP_ID_PK PRIMARY KEY(EMPNO),
CONSTRAINT EMP_MGR_FK FOREIGN KEY(MANAGERNO) REFERENCES
EMPLOYEE(EMPNO),
CONSTRAINT EMP_DEPTNO_FK FOREIGN KEY(DEPTNO) REFERENCES
DEPARTMENT(DEPTNO));

INSERT INTO EMPLOYEE VALUES(7839,'KING','PRESIDENT','','81-11-
17',5000,10);
INSERT INTO EMPLOYEE VALUES(7566,'JONES','MANAGER',7839,'81-02-
04',2975,20);
INSERT INTO EMPLOYEE VALUES(7902,'FORD','ANALYST',7566,'81-03-
12',3000,20);
```

```
SQL> CREATE TABLE EMPLOYEE
(EMPNO NUMBER(4) CONSTRAINT EMP_EMPNO_NN NOT NULL,
ENAME VARCHAR2(15) CONSTRAINT EMP_ENAME_NN NOT NULL,
JOB VARCHAR2(15),
MANAGERNO  NUMBER(4),
STARTDATE  DATE,
SALARY  NUMBER(10),
DEPTNO  NUMBER(3),
CONSTRAINT EMP_ID_PK PRIMARY KEY(EMPNO),
CONSTRAINT EMP_MGR_FK FOREIGN KEY(MANAGERNO) REFERENCES EMPLOYEE(EMPNO),
CONSTRAINT EMP_DEPTNO_FK FOREIGN KEY(DEPTNO) REFERENCES DEPARTMENT(DEPTNO));   2    3    4    5
  6    7    8    9   10   11

Table 'EMPLOYEE' created.

SQL> INSERT INTO EMPLOYEE VALUES(7839,'KING','PRESIDENT','','81-11-17',5000,10);

1 row inserted.

SQL> INSERT INTO EMPLOYEE VALUES(7566,'JONES','MANAGER',7839,'81-02-04',2975,20);

1 row inserted.

SQL> INSERT INTO EMPLOYEE VALUES(7902,'FORD','ANALYST',7566,'81-03-12',3000,20);

1 row inserted.
```

테이블 조회

SELECT * FROM DEPARTMENT;

```
SQL> SELECT * FROM DEPARTMENT;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

4 rows selected.
```

SET LINES 100

COL STARTDATE FOR A20

SELECT * FROM EMPLOYEE;

```
SQL> SET LINES 100
SQL> COL STARTDATE FOR A20
SQL> SELECT * FROM EMPLOYEE;

     EMPNO ENAME          JOB              MANAGERNO STARTDATE              SALARY     DEPTNO
---------- -------------- -------------- ---------- -------------------- ---------- ----------
      7566 JONES          MANAGER              7839 0081/02/04                 2975         20
      7839 KING           PRESIDENT                 0081/11/17                 5000         10
      7902 FORD           ANALYST              7566 0081/03/12                 3000         20

3 rows selected.
```

ROWID 출력

SELECT ROWID, DEPARTMENT.* FROM DEPARTMENT;

```
ROWID                  DEPTNO DNAME          LOC
------------------ ---------- -------------- -------------
AAAArDAACAAAABGAAA         10 ACCOUNTING     NEW YORK
AAAArDAACAAAABGAAB         20 RESEARCH       DALLAS
AAAArDAACAAAABGAAC         30 SALES          CHICAGO
AAAArDAACAAAABGAAD         40 OPERATIONS     BOSTON

4 rows selected.
```

ROWNUM 출력

```
SELECT ROWNUM, DEPARTMENT.* FROM DEPARTMENT;
```

```
    ROWNUM     DEPTNO DNAME          LOC
---------- ---------- -------------- -------------
         1         10 ACCOUNTING     NEW YORK
         2         20 RESEARCH       DALLAS
         3         30 SALES          CHICAGO
         4         40 OPERATIONS     BOSTON

4 rows selected.
```

```
SELECT * FROM DEPARTMENT WHERE ROWNUM<2;
```

```
SQL> SELECT * FROM DEPARTMENT WHERE ROWNUM<2;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK

1 row selected.
```

```
SELECT * FROM DEPARTMENT WHERE ROWNUM<1;
```

```
SQL> SELECT * FROM DEPARTMENT WHERE ROWNUM>1;

0 row selected.
```

결론 : 0row -> ROWNUM 값이 확정되기 전에 ROWNUM 에 대한 조건식이 수행되기 때문이다. 위의 SELECT 문의 결과는 첫 번째 로우가 ROWNUM = 1 이기 때문에 조건식을 만족하지 않는다. 조건식이 만족해야 반환된다.

2.5.3. LEVEL

- LEVEL 은 계층 질의를 실행한 결과에 각 로우의 트리 내 계층을 출력하기 위한 컬럼 타입이다.
- 최상위 로우의 LEVEL 값은 1 이며, 하위 로우로 갈수록 1 씩 증가한다.

2.5.4. CONNECT_BY_ISLEAF

- 현재 로우가 CONNECT BY 조건에 의해 정의된 트리(Tree)의 리프(Leaf)이면 1 을 반환하고 그렇지 않을 경우에는 0 을 반환한다.

2.5.5. CONNECT_BY_ISCYCLE

- 계층형 질의에서 사용되는 의사 컬럼으로서 해당 로우가 자식 노드를 갖고 있음과 동시에 그 자식 노드가 해당 로우의 부모 노드가 되는지를 판별한다.

시나리오 수행_LEVEL, CONNECT_BY_ISLEAF

시나리오 내용
TEST 사용자 접속
DEPARTMENT 테이블 생성 및 데이터 입력

시나리오 수행내역
TEST 사용자 접속
conn test/test
LEVEL 및 CONNECT_BY_ISLEAF 출력

```
/*START WITH 'KING'*/
SET LINES 100
COL PATH FOR A20
SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL,
SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
FROM EMPLOYEE
START WITH ENAME='KING'
CONNECT BY PRIOR EMPNO=MANAGERNO
ORDER BY ENAME;
```

```
SQL> SET LINES 100
SQL> COL PATH FOR A20
SQL> ^C
SQL> SELECT ENAME, CONNECT BY ISLEAF, LEVEL, SYS CONNECT BY PATH(ENAME,' ') "PATH"
  2 FROM EMPLOYEE
  3 START WITH ENAME='KING'
  4 CONNECT BY PRIOR EMPNO=MANAGERNO
  5 ORDER BY ENAME;

ENAME           CONNECT BY ISLEAF      LEVEL PATH
--------------- ----------------- ---------- --------------------
FORD                            1          3 -KING-JONES-FORD
JONES                           0          2 -KING-JONES
KING                            0          1 -KING

3 rows selected.
```

```
/*START WITH 'JONES'*/
SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL,
SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
FROM EMPLOYEE
START WITH ENAME='JONES'
CONNECT BY PRIOR EMPNO=MANAGERNO
ORDER BY ENAME;
```

```
SQL> SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL, SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
  2 FROM EMPLOYEE
  3 START WITH ENAME='JONES'
  4 CONNECT BY PRIOR EMPNO=MANAGERNO
  5 ORDER BY ENAME;

ENAME           CONNECT BY ISLEAF      LEVEL PATH
--------------- ----------------- ---------- --------------------
FORD                            1          2 -JONES-FORD
JONES                           0          1 -JONES

2 rows selected.
```

```
/*START WITH 'FORD'*/
SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL,
SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
FROM EMPLOYEE
START WITH ENAME='FORD'
```

결론 : JONES 의 MANAGER 는 KING, FORD 의 MANAGER 는 JONES, KING 의 MANAGER 는 없다.

2.6. NULL

2.6.1. 함수에서의 NULL

- REPLACE, NVL, CONCAT 을 제외한 모든 상수 함수는 함수의 파라미터가 NULL 일 경우 반환 값은 NULL 이다.

2.6.1. NULL 에 대한 비교조건

- IS NULL 과 IS NOT NULL 만 가능하다.
- 때문에 NULL 과 NULL, NULL 과 NULL 이 아닌 다른 값을 서로 비교할 수 없다.

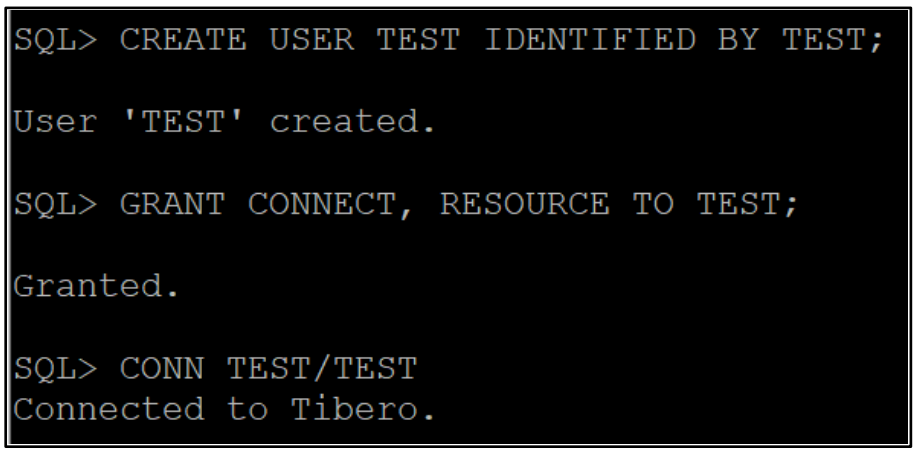
잘못된 사용
① **job = NULL** (NULL 은 비교 연산자를 사용할 수 없다) ② **job != NULL** (NULL 은 비교 연산자를 사용할 수 없다) ③ **job = ''** (빈 문자열은 비교 연산자를 사용할 수 없다) ④ **sal + NULL** (수치값에 NULL 을 사칙연산하면 결과는 NULL 이다)

올바른 사용
① **job IS NULL** (NULL 을 조건으로 사용할 때는 IS NULL 을 사용한다) ② **job IS NOT NULL** (NULL 을 조건으로 사용할 때는 IS NOT NULL 을 사용한다) ③ **job IS NULL** (빈 문자열은 NULL 을 사용한다) ④ **sal + NVL(NULL, 0)** (수치값 또는 컬럼은 NULL 이 존재할 경우 NVL 로 치환한다)

시나리오 수행_NULL

시나리오 내용
TEST 접속

데이터 입력
IS NULL, =NULL
IS NOT NULL, !=NULL
NULL 값과의 연산
문자열 비교연산자
IN, NOT IN
DECODE 함수
CASE 함수

시나리오 수행내역
TEST 사용자 생성
CREATE USER TEST IDENTIFIED BY TEST; GRANT CONNECT, GRANT TO TEST; CONN TEST/TEST  SQL> CREATE USER TEST IDENTIFIED BY TEST; User 'TEST' created. SQL> GRANT CONNECT, RESOURCE TO TEST; Granted. SQL> CONN TEST/TEST Connected to Tibero.
EMPLOYEE 데이터 추가

```
INSERT INTO EMPLOYEE VALUES(7698,'BLAKE','MANAGER',7839,'81-05-
01',2850,30);
INSERT INTO EMPLOYEE VALUES(7782,'CLARK','MANAGER',7839,'81-05-
09',2450,10);
INSERT INTO EMPLOYEE VALUES(7566,'JONES','MANAGER',7839,'81-04-
01',2975,20);
INSERT INTO EMPLOYEE VALUES(7654,'MARTIN','SALESMAN',7698,'81-02-
11',1250,30);
INSERT INTO EMPLOYEE VALUES(7499,'ALLEN','SALESMAN',7839,'81-09-
10',1600,30);
INSERT INTO EMPLOYEE VALUES(7844,'TURNER','SALESMAN',7698,'81-08-
21',1500,30);
INSERT INTO EMPLOYEE VALUES(7900,'JAMES','CLERK',7698,'81-12-
11',950,30);
INSERT INTO EMPLOYEE VALUES(7521,'WARD','SALESMAN',7698,'81-02-
23',1250,30);
INSERT INTO EMPLOYEE VALUES(7369,'SMITH','CLERK',7902,'80-12-
09',800,20);
INSERT INTO EMPLOYEE VALUES(7788,'SCOTT','ANALTST',7566,'82-12-
22',3000,20);
INSERT INTO EMPLOYEE VALUES(7876,'ADAMS','CLERK',7788,'83-01-
15',1100,20);
INSERT INTO EMPLOYEE VALUES(7934,'MILLER','CLERK',7782,'82-01-
11',1300,10);
```

```
IS NULL,  =NULL
```

SELECT * FROM EMPLOYEE WHERE MANAGERNO IS NULL;

SELECT * FROM EMPLOYEE WHERE MANAGERNO = NULL;

```
SQL> SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP
---------------------------------------------------------------------------
2022/11/09 17:04:05.430976 Asia/Seoul

1 row selected.

SQL> SELECT * FROM EMPLOYEE WHERE MANAGERNO IS NULL;

     EMPNO ENAME           JOB              MANAGERNO STARTDATE      SALARY     DEPTNO

      7839 KING            PRESIDENT                  0081/11/17       5000         10

1 row selected.

SQL> SELECT * FROM EMPLOYEE WHERE MANAGERNO=NULL;

0 row selected.
```

IS NOT NULL, !=NULL

SELECT * FROM EMPLOYEE WHERE MANAGERNO IS NOT NULL;

SELECT * FROM EMPLOYEE WHERE MANAGERNO != NULL;

```
SQL> SELECT * FROM EMPLOYEE WHERE MANAGERNO IS NOT NULL;

     EMPNO ENAME           JOB              MANAGERNO STARTDATE      SALARY     DEPTNO
---------- --------------- --------------- ---------- ---------- ---------- ----------
      7369 SMITH           CLERK                 7902 0080/12/09        800         20
      7499 ALLEN           SALESMAN              7839 0081/09/10       1600         30
      7521 WARD            SALESMAN              7698 0081/02/23       1250         30
      7566 JONES           MANAGER               7839 0081/02/04       2975         20
      7654 MARTIN          SALESMAN              7698 0081/02/11       1250         30
      7698 BLAKE           MANAGER               7839 0081/05/01       2850         30
      7782 CLARK           MANAGER               7839 0081/05/09       2450         10
      7788 SCOTT           ANALYST               7566 0082/12/22       3000         20
      7844 TURNER          SALESMAN              7698 0081/08/21       1500         30
      7876 ADAMS           CLERK                 7788 0083/01/15       1100         20
      7900 JAMES           CLERK                 7698 0081/12/11        950         30
      7902 FORD            ANALYST               7566 0081/03/12       3000         20
      7990 NEW             ANALYST               7902 0081/05/12       2500         30

13 rows selected.

SQL> SELECT * FROM EMPLOYEE WHERE MANAGERNO!=NULL;

0 row selected.
```

NULL 값과의 연산(NVL())

SELECT * FROM EMPLOYEE WHERE NVL(MANAGERNO,0)!=7839;

```
SQL> SELECT * FROM EMPLOYEE WHERE NVL(MANAGERNO,0)!=7839
  2 ;

     EMPNO ENAME      JOB              MANAGERNO STARTDATE      SALARY     DEPTNO

      7369 SMITH      CLERK                7902 0080/12/09        800         20
      7521 WARD       SALESMAN             7698 0081/02/23       1250         30
      7654 MARTIN     SALESMAN             7698 0081/02/11       1250         30
      7788 SCOTT      ANALYST              7566 0082/12/22       3000         20
      7844 TURNER     SALESMAN             7698 0081/08/21       1500         30
      7876 ADAMS      CLERK                7788 0083/01/15       1100         20
      7900 JAMES      CLERK                7698 0081/12/11        950         30
      7902 FORD       ANALYST              7566 0081/03/12       3000         20
      7990 NEW        ANALYST              7902 0081/05/12       2500         30
      7839 KING       PRESIDENT                 0081/11/17       5000         10

10 rows selected.
```

결론 : NVL 함수 이용하여 비교하면 조건에 맞는 칼럼중 NULL 값도 포함

IN, NOT IN

SELECT * FROM EMPLOYEE WHERE NVL(MANAGERNO,0) IN 7698;

SELECT * FROM EMPLOYEE WHERE NVL(MANAGERNO,0) NOT IN 7698;

```
SQL> SELECT * FROM EMPLOYEE WHERE NVL(MANAGERNO,0) IN 7698;

     EMPNO ENAME      JOB              MANAGERNO STARTDATE      SALARY     DEPTNO

      7521 WARD       SALESMAN             7698 0081/02/23       1250         30
      7654 MARTIN     SALESMAN             7698 0081/02/11       1250         30
      7844 TURNER     SALESMAN             7698 0081/08/21       1500         30
      7900 JAMES      CLERK                7698 0081/12/11        950         30

4 rows selected.

SQL> SELECT * FROM EMPLOYEE WHERE NVL(MANAGERNO,0) NOT IN 7698;

     EMPNO ENAME      JOB              MANAGERNO STARTDATE      SALARY     DEPTNO
---------- ---------- --------------- ---------- ---------- ---------- ----------
      7369 SMITH      CLERK                7902 0080/12/09        800         20
      7499 ALLEN      SALESMAN             7839 0081/09/10       1600         30
      7566 JONES      MANAGER              7839 0081/02/04       2975         20
      7698 BLAKE      MANAGER              7839 0081/05/01       2850         30
      7782 CLARK      MANAGER              7839 0081/05/09       2450         10
      7788 SCOTT      ANALYST              7566 0082/12/22       3000         20
      7876 ADAMS      CLERK                7788 0083/01/15       1100         20
      7902 FORD       ANALYST              7566 0081/03/12       3000         20
      7990 NEW        ANALYST              7902 0081/05/12       2500         30
      7839 KING       PRESIDENT                 0081/11/17       5000         10

10 rows selected.
```

DECODE 함수

```
SELECT EMPNO,ENAME,MANAGERNO,DECODE(MANAGERNO,NULL,1,0) NOMANAGER
FROM EMPLOYEE;
```

```
SQL> SELECT EMPNO,ENAME,MANAGERNO,DECODE(MANAGERNO,NULL,1,0) NOMANAGER FROM EMPLOYEE;

     EMPNO ENAME           MANAGERNO  NOMANAGER
---------- --------------- ---------- ----------
      7839 KING                             1
      7566 JONES                 7839         0
      7902 FORD                  7566         0
      7990 NEW                   7902         0
      7698 BLAKE                 7839         0
      7782 CLARK                 7839         0
      7654 MARTIN                7698         0
      7499 ALLEN                 7839         0
      7844 TURNER                7698         0
      7521 WARD                  7698         0
      7369 SMITH                 7902         0
      7900 JAMES                 7698         0
      7788 SCOTT                 7566         0
      7876 ADAMS                 7788         0

14 rows selected.
```

CASE 함수 (DECODE 함수때와 같은 결과)

```
SELECT EMPNO,ENAME,MANAGERNO,CASE WHEN MANAGERNO IS NULL THEN 1
ELSE 0 END NOMANAGER FROM EMPLOYEE;
```

```
SQL> SELECT EMPNO,ENAME,MANAGERNO,CASE WHEN MANAGERNO IS NULL THEN 1 ELSE 0 END NOMANAGER FROM EMPLOYEE;

     EMPNO ENAME           MANAGERNO  NOMANAGER

      7369 SMITH                 7902         0
      7499 ALLEN                 7839         0
      7521 WARD                  7698         0
      7566 JONES                 7839         0
      7654 MARTIN                7698         0
      7698 BLAKE                 7839         0
      7782 CLARK                 7839         0
      7788 SCOTT                 7566         0
      7839 KING                               1
      7844 TURNER                7698         0
      7876 ADAMS                 7788         0
      7900 JAMES                 7698         0
      7902 FORD                  7566         0
      7990 NEW                   7902         0

14 rows selected.
```

2.7. 주석

- 주석은 SQL 문장의 실행에는 전혀 영향을 주지 않는다.
- 방법
  - 시작 기호(/*)로 주석의 시작을 나타내고 마침 기호(*/)로 주석을 끝낸다.
  - 주석의 내용을 여러 줄에 걸쳐 삽입할 수 있다. 시작 기호(/*)와 마침 기호(*/)를 내용과 구분하기 위해 공백이나 줄 바꿈을 사용할 필요는 없다.

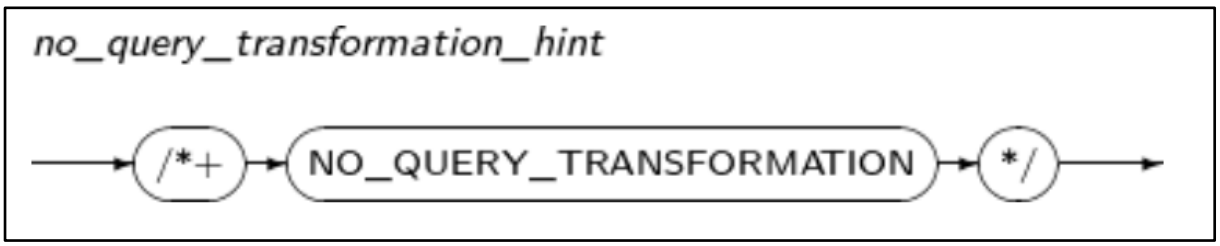
- '--'로 주석의 시작을 나타내고 바로 뒤에 주석의 내용을 적는다.
- 해당 줄의 끝이 주석의 끝을 나타내므로 주석의 내용이 다음 줄로 넘어가서는 안 된다.

2.8. 힌트

2.8.1. 질의 변형

NO_QUERY_TRANSFORMATION

- 질의 변형기(Query Transformer)가 전체 쿼리에 대해 변형을 실행하지 않도록 지시하는 힌트이다.
- 문법



시나리오 수행_NO_QUERY_TRANSFORMATION

시나리오 내용
T1.T2 테이블 생성
원본 테이블 조회
NO_QUERY_TRANSFORMATION HINT 수행

시나리오 수행내역
TEST 사용자 접속
conn test/test
LEVEL 및 CONNECT_BY_ISLEAF 출력

```
/*START WITH 'KING'*/
SET LINES 100
COL PATH FOR A20
SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL,
SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
FROM EMPLOYEE
START WITH ENAME='KING'
CONNECT BY PRIOR EMPNO=MANAGERNO
ORDER BY ENAME;
```

```
SQL> SET LINES 100
SQL> COL PATH FOR A20
SQL> ^C
SQL> SELECT ENAME, CONNECT BY ISLEAF, LEVEL, SYS CONNECT BY PATH(ENAME,' ') "PATH"
  2 FROM EMPLOYEE
  3 START WITH ENAME='KING'
  4 CONNECT BY PRIOR EMPNO=MANAGERNO
  5 ORDER BY ENAME;

ENAME           CONNECT BY ISLEAF      LEVEL PATH
--------------- ----------------- ---------- --------------------
FORD                            1          3 -KING-JONES-FORD
JONES                           0          2 -KING-JONES
KING                            0          1 -KING

3 rows selected.
```

```
/*START WITH 'JONES'*/
SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL,
SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
FROM EMPLOYEE
START WITH ENAME='JONES'
CONNECT BY PRIOR EMPNO=MANAGERNO
ORDER BY ENAME;
```

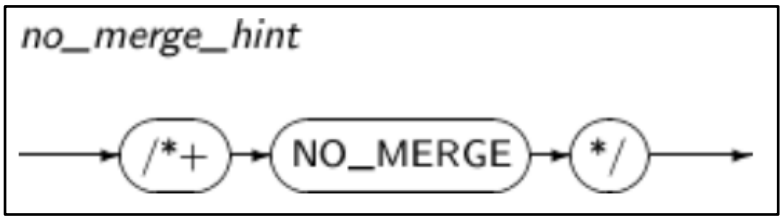
```
SQL> SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL, SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
  2 FROM EMPLOYEE
  3 START WITH ENAME='JONES'
  4 CONNECT BY PRIOR EMPNO=MANAGERNO
  5 ORDER BY ENAME;

ENAME           CONNECT BY ISLEAF      LEVEL PATH
--------------- ----------------- ---------- --------------------
FORD                            1          2 -JONES-FORD
JONES                           0          1 -JONES

2 rows selected.
```

```
/*START WITH 'FORD'*/
SELECT ENAME, CONNECT_BY_ISLEAF, LEVEL,
SYS_CONNECT_BY_PATH(ENAME,'-') "PATH"
FROM EMPLOYEE
START WITH ENAME='FORD'
```

결론 : JONES 의 MANAGER 는 KING, FORD 의 MANAGER 는 JONES, KING 의 MANAGER 는 없다.

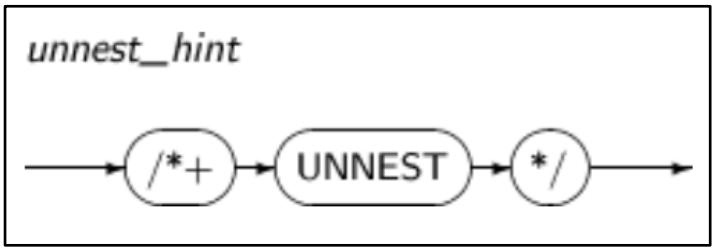
NO_MERGE

- 질의 변형기(Query Transformer)가 특정 뷰에 대해 뷰 병합을 하지 않도록 지시하는 힌트이다.
- Tibero 에서는 뷰 병합이 디폴트로 수행되며, 뷰가 병합이 가능할 경우 상위의 질의 블록과 결합해 하나의 질의 블록을 형성한다.
- 문법



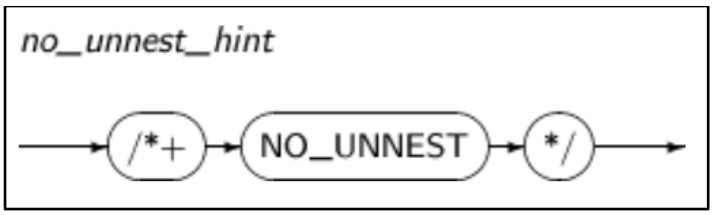
UNNEST

- 질의 변형기가 특정 부질의(Subquery)를 언네스팅하도록 지시하는 힌트이다.
- 특정 쿼리만 언네스팅을 하려면 초기화 파라미터에서 언네스팅을 해제하면 된다.
- 문법



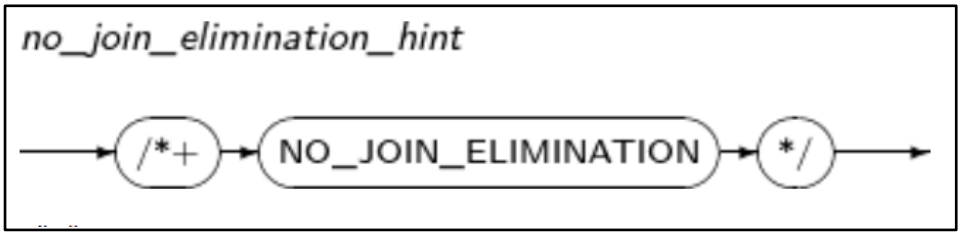
NO_UNNEST

- 질의 변형기가 특정 부질의에 대해 언네스팅을 수행하지 않도록 지시하는 힌트이다.
- Tibero 는 부질의 언네스팅을 디폴트로 수행하며 언네스팅이 가능한 경우 부질의를 조인으로 변환한다.
- 문법



NO_JOIN_ELIMINATION

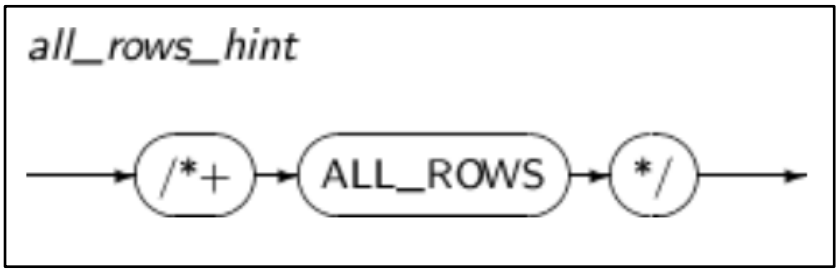
- 질의 변형기(Query Transformer)가 불필요한 조인을 찾아서 제거하지 않도록 지시하는 힌트이다.
- 문법



2.8.2. 최적화 방법

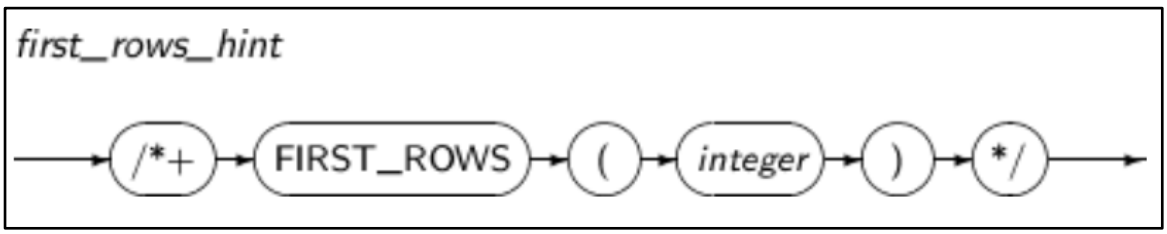
ALL_ROWS

- 최소한의 리소스를 사용하여 전체 결과에 대한 처리량이 가장 많도록 처리과정의 최적화 방법을 선택하는 힌트이다.
- 문법



FIRST_ROWS

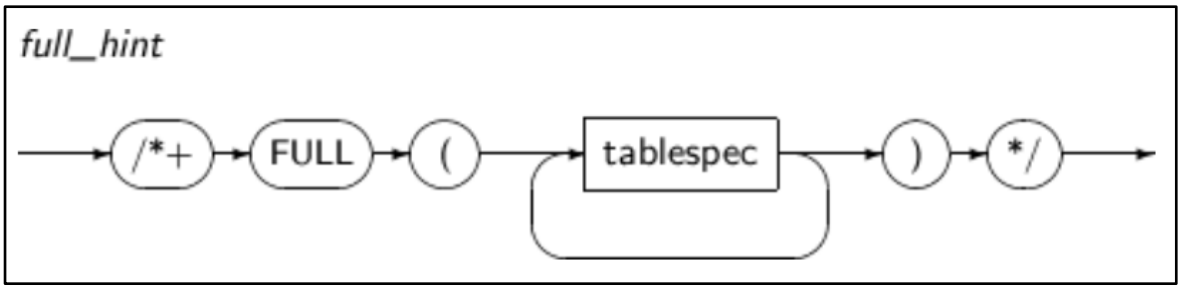
- 첫 로우부터 파라미터로 입력된 번호의 로우까지 가장 빠르게 보여줄 수 있도록 결과 표시의 최적화 방법을 선택하는 힌트이다.
- 문법



2.8.3. 접근 방법

FULL

- 명시한 테이블을 스캔할 때, 전체 테이블을 스캔하도록 지시하는 힌트이다.
- 문법



시나리오 수행_HINT(FULL)

시나리오 내용
고객 테이블 생성 (HINT_TEST1)
INDEX 생성
고객 상세 테이블 생성 (HINT_TEST2)
INDEX 생성
INDEX HINT 사용
FULL HINT 사용
TRACE 확인

시나리오 수행내역
고객 테이블 생성 (HINT_TEST1)

```
create table HINT_TEST1 (
name varchar(20),
sex varchar(1)
) TABLESPACE HINT_TEST_SPACE;

insert into HINT_TEST1 select 'HUMAN'||to_char(lpad(level,8,'0')),
TO_CHAR(ROUND(DBMS_RANDOM.VALUE(1,2), 0)) from
dual connect by level <= 1000;
```

```
SQL> create table HINT_TEST1 (
  2 name varchar(20),
  3 sex varchar(1)
  4 ) TABLESPACE HINT_TEST_SPACE;

Table 'HINT_TEST1' created.

SQL>
SQL> insert into HINT_TEST1 select 'HUMAN'||to_char(lpad(level,8,'0')), TO_CHAR(
ROUND(DBMS_RANDOM.VALUE(1,2), 0)) from
  2 dual connect by level <= 1000;

1000 rows inserted.
```

INDEX 생성

```
create index IDX_01_HINT_TEST1 ON HINT_TEST1(sex) TABLESPACE
HINT_TEST_SPACE_IDX;
create index IDX_02_HINT_TEST1 ON HINT_TEST1(name, sex) TABLESPACE
HINT_TEST_SPACE_IDX;
```

```
SQL> create index IDX_01_HINT_TEST1 ON HINT_TEST1(sex) TABLESPACE HINT_TEST_SPAC
E_IDX;

Index 'IDX_01_HINT_TEST1' created.

SQL> create index IDX_02_HINT_TEST1 ON HINT_TEST1(name, sex) TABLESPACE HINT_TES
T_SPACE_IDX;

Index 'IDX_02_HINT_TEST1' created.
```

고객 상세 테이블 생성 (HINT_TEST2)

```
create table HINT_TEST2 (
name varchar(20),
addr varchar(50),
phone varchar(20)
) TABLESPACE HINT_TEST_SPACE;

insert into HINT_TEST2 select 'HUMAN'||to_char(lpad(level,8,'0')),
'ADDR-'||DBMS_RANDOM.STRING('U',20), '010-1111-2222'
from dual connect by level <= 1000;
commit;
```

```
SQL> create table HINT_TEST2 (
  2 name varchar(20),
  3 addr varchar(50),
  4 phone varchar(20)
  5 ) TABLESPACE HINT_TEST_SPACE;

Table 'HINT_TEST2' created.

SQL>
SQL> insert into HINT_TEST2 select 'HUMAN'||to_char(lpad(level,8,'0')), 'ADDR-'|
|DBMS_RANDOM.STRING('U',20), '010-1111-2222'
  2 from dual connect by level <= 1000;

1000 rows inserted.

SQL> commit;

Commit completed.
```

인덱스 생성

```
create index IDX_01_HINT_TEST2 on HINT_TEST2(name) TABLESPACE
HINT_TEST_SPACE_IDX;
```

```
SQL> create index IDX_01_HINT_TEST2 on HINT_TEST2(name) TABLESPACE HINT_TEST_SPACE_IDX;

Index 'IDX_01_HINT_TEST2' created.
```

INDEX HINT 사용 & TRACE 확인

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT

SELECT /*+INDEX(B IDX_01_HINT_TEST1)*/ B.NAME,B.PHONE
FROM (SELECT /*+INDEX(A1 IDX_01_HINT_TEST1)*/ --TEST1
/*+INDEX(A1 IDX_01_HINT_TEST2)*/ --TEST2
A1.NAME
FROM HINT_TEST1 A1 WHERE A1.SEX='1')A, HINT_TEST2 B
WHERE B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774')
AND B.NAME=A.NAME;
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:7, %%CPU:0, Rows:2)
   2    TABLE ACCESS (ROWID): HINT_TEST1 (Cost:5, %%CPU:0, Rows:2)
   3      INDEX (RANGE SCAN): IDX_01_HINT_TEST1 (Cost:2, %%CPU:0, Rows:499)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   5      FILTER (Cost:2, %%CPU:0, Rows:1)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Cost:2, %%CPU:0, Rows:1)


Predicate Information
----------------------------------------------------------------------------------------------------
   2   filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   3 - access: ("A1"."SEX" = '1') (0.499)
   5   filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   6 - access: ("B"."NAME" = "A1"."NAME") (0.200)


Note

   6 - dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Time:0. ms, Rows:0, Starts:0)
   2    TABLE ACCESS (ROWID): HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (RANGE SCAN): IDX_01_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   5      FILTER (Time:0. ms, Rows:0, Starts:0)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
```

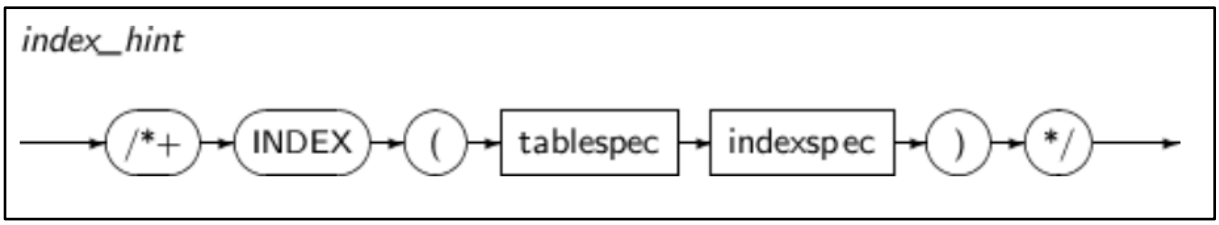
FULL HINT 사용 & TRACE 확인

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT

SELECT /*+INDEX(B IDX_01_HINT_TEST1)*/ B.NAME,B.PHONE
FROM (SELECT /*+INDEX(A1 IDX_01_HINT_TEST1)*/ --TEST1
/*+FULL(A1)*/ --TEST2
A1.NAME
FROM HINT_TEST1 A1 WHERE A1.SEX='1')A, HINT_TEST2 B
WHERE B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774')
AND B.NAME=A.NAME;
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:7, %%CPU:0, Rows:2)
   2    FILTER (Cost:5, %%CPU:0, Rows:2)
   3      INDEX (FULL): IDX_02_HINT_TEST1 (Cost:5, %%CPU:0, Rows:1000)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   5      FILTER (Cost:2, %%CPU:0, Rows:1)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Cost:2, %%CPU:0, Rows:1)


Predicate Information

   2 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) AND ("A1"."SEX" = '1') (0.005 * 0.499)

   5 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   6   access: ("B"."NAME" = "A1"."NAME") (0.200)
```

INDEX

- 명시한 테이블을 스캔할 때, 명시한 인덱스를 사용하여 인덱스 스캔을 하도록 지시하는 힌트이다.
- 문법



시나리오 수행_HINT INDEX

시나리오 내용
INDEX 힌트 사용 방법

TRACE 비교

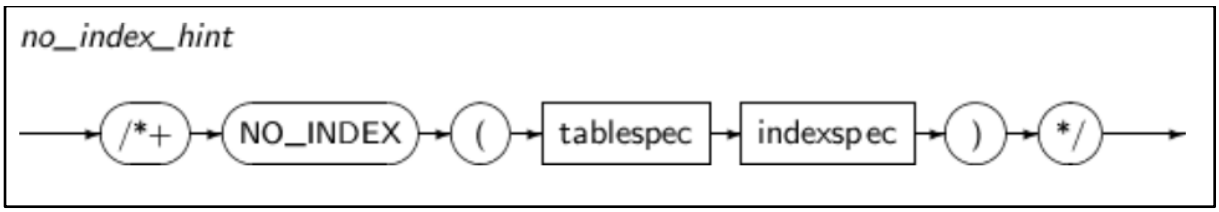
시나리오 수행내역
INDEX 힌트 사용 방법 & TRACE 확인

```
--ALIAS 사용
SELECT /*+ INDEX(A1 IDX_01_HINT_TEST11) */
 A1.NAME
FROM HINT_TEST1 A1 WHERE A1.SEX = '1'
--TABLENAME 사용
SELECT /*+ INDEX(HINT_TEST1 IDX_01_HINT_TEST1) */
 NAME
FROM HINT_TEST1 WHERE SEX = '1';
```

```
Predicate Information

----------------------
   2 - access: ("A1"."SEX" = '1') (0.499)


Execution Stat
-------------------------------------------------------------------------------
----------------------
   1  TABLE ACCESS (ROWID): HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (RANGE SCAN): IDX_01_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)

SQL> SELECT /*+ INDEX(HINT_TEST1 IDX_01_HINT_TEST1) */
 NAME
FROM HINT_TEST1 WHERE SEX = '1';   2    3

SQL_ID: 9w00x64zc3xdd
Child number: 1006
Plan hash value: 4010878762

Execution Plan
-------------------------------------------------------------------------------------------------------------
   1  TABLE ACCESS (ROWID): HINT_TEST1 (Cost:5, %%CPU:0, Rows:499)
   2    INDEX (RANGE SCAN): IDX_01_HINT_TEST1 (Cost:2, %%CPU:0, Rows:499)


Predicate Information

   2   access: ("HINT_TEST1"."SEX" = '1') (0.499)


Execution Stat

   1  TABLE ACCESS (ROWID): HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (RANGE SCAN): IDX_01_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
```

NO_INDEX

- 명시한 테이블을 스캔할 때, 명시한 인덱스를 사용하는 인덱스 스캔을 하지 않도록 지시하는 힌트이다.
- 문법



시나리오 수행_HINT NO_INDEX

시나리오 내용
NO_INDEX HINT 사용
TRACE 확인

시나리오 수행내역
NO_INDEX HINT 사용

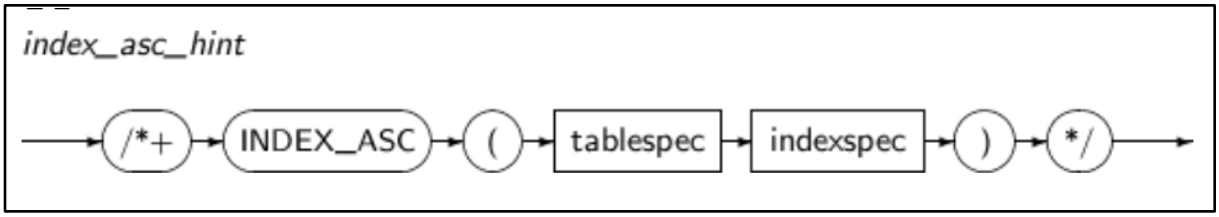
```
SELECT /*+ INDEX(B IDX_01_HINT_TEST2) */
 B.NAME, B.PHONE
FROM ( SELECT /*+ NO_INDEX(A1 IDX_01_HINT_TEST1) */
 A1.NAME
 FROM HINT_TEST1 A1 WHERE A1.SEX = '1') A,
 HINT_TEST2 B
WHERE B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774')
AND B.NAME = A.NAME;
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:9, %%CPU:0, Rows:2)
   2    FILTER (Cost:6, %%CPU:0, Rows:2)
   3      INDEX (FAST FULL SCAN): IDX_02_HINT_TEST1 (Cost:6, %%CPU:0, Rows:1000)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   5      FILTER (Cost:2, %%CPU:0, Rows:1)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Cost:2, %%CPU:0, Rows:1)


Predicate Information
----------------------------------------------------------------------------------------------------
   2 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) AND ("A1"."SEX" = '1') (0.005 * 0.499)

   5 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   6 - access: ("B"."NAME" = "A1"."NAME") (0.200)


Note
----------------------------------------------------------------------------------------------------
   6 - dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Time:0. ms, Rows:0, Starts:0)
   2    FILTER (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (FAST FULL SCAN): IDX_02_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   5      FILTER (Time:0. ms, Rows:0, Starts:0)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
```

결론 : HINT_TEST1 테이블에는 IDX_01_HINT_TEST1 이라는 INDEX 하나만 있기 때문에 FULL 로 수행되었다.

INDEX_ASC

- INDEX 와 동일한 작업 수행
- 문법



시나리오 수행_INDEX_ASC

시나리오 내용
HINT TEST3 테이블 및 인덱스 생성
IDX_01_HINT_TEST3 로 MIN 함수와 INDEX_ASC HINT 와 비교
TRACE 확인
추가 INDEX 생성 후 MIN 함수와 INDEX_ASC HINT 와 비교
TRACE 확인

시나리오 수행내역
HINT TEST3 테이블 및 인덱스 생성

```
CREATE TABLE HINT_TEST3 (
NAME VARCHAR(20),
AGE VARCHAR(3),
SEL NUMBER,
ETC VARCHAR(50)
) TABLESPACE HINT_TEST_SPACE;

INSERT INTO HINT_TEST3 SELECT 'A123',
LPAD(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(1,100), 0)),3,0),
ROUND(DBMS_RANDOM.VALUE(1,1000), 0), 'TEST1' FROM DUAL CONNECT BY
LEVEL <= 500;

INSERT INTO HINT_TEST3 SELECT 'A223',
LPAD(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(1,100), 0)),3,0),
ROUND(DBMS_RANDOM.VALUE(1,1000), 0), 'TEST2' FROM DUAL CONNECT BY
LEVEL <= 500;

INSERT INTO HINT_TEST3 SELECT 'A323',
LPAD(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(1,100), 0)),3,0),
ROUND(DBMS_RANDOM.VALUE(1,1000), 0), 'TEST3' FROM DUAL CONNECT BY
LEVEL <= 500;

CREATE INDEX IDX_01_TIBERO_TEST3 ON HINT_TEST3 (SEL, NAME)
TABLESPACE HINT_TEST_SPACE_IDX;
CREATE INDEX IDX_01_TIBERO_TEST3 ON HINT_TEST3 (SEL, NAME, AGE)
TABLESPACE HINT_TEST_SPACE_IDX;
```

```
SQL> DESC HINT_TEST3;

COLUMN NAME                                       TYPE              CONSTRAINT
------------------------------------------------- ----------------- ----------------------------
NAME                                              VARCHAR(20)
AGE                                               VARCHAR(3)
SEL                                               NUMBER
ETC                                               VARCHAR(50)


INDEX_NAME                          TYPE                    COLUMN_NAME
----------------------------------- ----------------------- --------------------------------
IDX_01_TIBERO_TEST3                 NORMAL                  SEL
                                                            NAME
```

MIN 함수와 INDEX_ASC HINT 와 비교

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


SELECT MIN(A.SEL)
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60';
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  COLUMN PROJECTION (Cost:520, %%CPU:0, Rows:1)
   2    SORT AGGR (Cost:520, %%CPU:0, Rows:1)
   3      COUNT (STOP NODE) (STOP LIMIT 2) (Cost:520, %%CPU:0, Rows:1)
   4        TABLE ACCESS (ROWID): HINT_TEST3 (Cost:520, %%CPU:0, Rows:750)
   5          FILTER (Cost:7, %%CPU:0, Rows:750)
   6            INDEX (FULL): IDX_01_TIBERO_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------------------------------
   3 - filter: ("A"."SEL" IS NOT NULL) (1.000)
   4   filter: ("A"."AGE" < '60') (1.000)
   5   filter: ("A"."NAME" = 'A123') (0.312)


Note

   6   dynamic sampling used for this table (32 blocks)


Execution Stat

   1  COLUMN PROJECTION (Time:0. ms, Rows:0, Starts:0)
   2    SORT AGGR (Time:0. ms, Rows:0, Starts:0)
   3      COUNT (STOP NODE) (STOP LIMIT 2) (Time:0. ms, Rows:0, Starts:0)
   4        TABLE ACCESS (ROWID): HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
   5          FILTER (Time:0. ms, Rows:0, Starts:0)
   6            INDEX (FULL): IDX_01_TIBERO_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

```
SELECT /*+ INDEX_ASC(A IDX_01_HINT_TEST3) */ A.SEL
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60'
AND ROWNUM = 1;
```

```
Execution Plan

   1  COUNT (STOP NODE) (STOP LIMIT 2) (Cost:1272, %%CPU:0, Rows:1)
   2    FILTER (Cost:1272, %%CPU:0, Rows:750)
   3      TABLE ACCESS (ROWID): HINT_TEST3 (Cost:1272, %%CPU:0, Rows:750)
   4        INDEX (SKIP SCAN): IDX_01_TIBERO_TEST3 (Cost:750, %%CPU:0, Rows:750)


Predicate Information

   1   filter: (ROWNUM = 1) (0.010)
   2 - filter: ("A"."AGE" < '60') (1.000)
   4 - access: ("A"."NAME" = 'A123') (0.312)


Note

   4   dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------------------
   1  COUNT (STOP NODE) (STOP LIMIT 2) (Time:0. ms, Rows:0, Starts:0)
   2    FILTER (Time:0. ms, Rows:0, Starts:0)
```

• INDEX_ASC 힌트를 사용하여 SORT 는 없어졌지만 성능의 차이는 없다. • MIN & INDEX_ASC 사용 모두 IDX_01_TIBERO_TEST3 의 INDEX 가 효과적이지 않아 COST 가 많이 발생한다.
추가 INDEX 생성 후 MIN 함수와 INDEX_ASC HINT 와 비교

```
CREATE INDEX IDX_02_HINT_TEST3 ON HINT_TEST3 (SEL, NAME, AGE);
```

```
SQL> CREATE INDEX IDX_02_HINT_TEST3 ON HINT_TEST3 (SEL, NAME, AGE);

Index 'IDX_02_HINT_TEST3' created.
```

```
-- INDEX 추가 생성 후 해당 INDEX로 INDEX_ASC 힌트 사용
SELECT /*+ INDEX_ASC(A IDX_02_HINT_TEST3) */ A.SEL
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60'
AND ROWNUM = 1;
```

```
Execution Plan
----------------------------------------------------------------------------

   1  COUNT (STOP NODE) (STOP LIMIT 2) (Cost:7, %%CPU:0, Rows:1)
   2    FILTER (Cost:7, %%CPU:0, Rows:750)
   3      INDEX (FULL): IDX_02_HINT_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information

----------------------
   1 - filter: (ROWNUM = 1) (0.010)
   2 - filter: ("A"."NAME" = 'A123') AND ("A"."AGE" < '60') (0.312 * 1.000)


Note
----------------------------------------------------------------------------
----------------------
   3 - dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------
----------------------
   1  COUNT (STOP NODE) (STOP LIMIT 2) (Time:0. ms, Rows:0, Starts:0)
   2    FILTER (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (FULL): IDX_02_HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

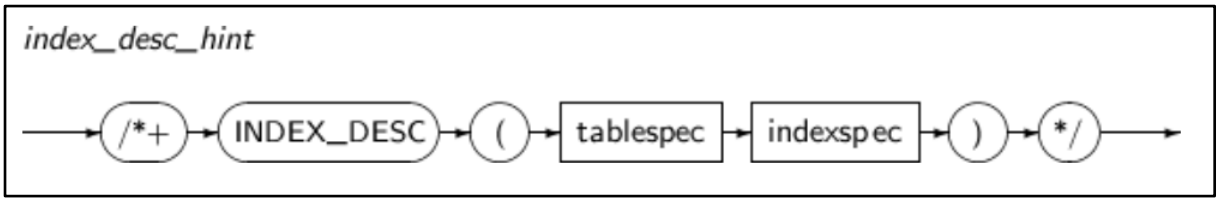
```
-- INDEX 추가 생성 후 해당 INDEX로 MIN 함수 사용
SELECT MIN(A.SEL)
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60';
```

```
Execution Plan
----------------------------------------------------------------------------
----------------------
   1  COLUMN PROJECTION (Cost:7, %%CPU:0, Rows:1)
   2    SORT AGGR (Cost:7, %%CPU:0, Rows:1)
   3      COUNT (STOP NODE) (STOP LIMIT 2) (Cost:7, %%CPU:0, Rows:1)
   4        FILTER (Cost:7, %%CPU:0, Rows:750)
   5          INDEX (FULL): IDX_02_HINT_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information
```

IDX_02_HINT_TEST3 은 출력부 및 조건부에 해당하는 컬럼(SEL, NAME, AGE)이 포함된 결합인덱스로 INDEX 만 읽어서 한번에 FILTER 되어 실행계획이 단순해 졌으며 비용이 많이 소요된 TABLE ACCESS 가 제거 되었다.

INDEX_DESC

- 인덱스를 내림차순으로 스캔하도록 한다.
- 문법



시나리오 수행_INDEX_DESC

시나리오 내용
IDX_01_HINT_TEST3 로 MAX 함수와 INDEX_DESC HINT 와 비교
TRACE 확인
추가 INDEX 생성 후 MAX 함수와 INDEX_DESC HINT 와 비교
TRACE 확인

시나리오 수행내역
MIN 함수와 INDEX_ASC HINT 와 비교

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT

SELECT MAX(A.SEL)
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60';
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------
   1  COLUMN PROJECTION (Cost:519, %%CPU:0, Rows:1)
   2    SORT AGGR (Cost:519, %%CPU:0, Rows:1)
   3      COUNT (STOP NODE) (STOP LIMIT 2) (Cost:519, %%CPU:0, Rows:1)
   4        TABLE ACCESS (ROWID): HINT_TEST3 (Cost:519, %%CPU:0, Rows:750)
   5          FILTER (Cost:7, %%CPU:0, Rows:750)
   6            INDEX (FULL) DESCENDING: IDX_01_TIBERO_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information

----------------------------------------------------------------
   3   filter: ("A"."SEL" IS NOT NULL) (1.000)
   4 - filter: ("A"."AGE" < '60') (1.000)
   5 - filter: ("A"."NAME" = 'A123') (0.312)


Note
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------
   6 - dynamic sampling used for this table (32 blocks)


Execution Stat

----------------------------------------------------------------
   1  COLUMN PROJECTION (Time:0. ms, Rows:0, Starts:0)
   2    SORT AGGR (Time:0. ms, Rows:0, Starts:0)
   3      COUNT (STOP NODE) (STOP LIMIT 2) (Time:0. ms, Rows:0, Starts:0)
   4        TABLE ACCESS (ROWID): HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
   5          FILTER (Time:0. ms, Rows:0, Starts:0)
   6            INDEX (FULL) DESCENDING: IDX_01_TIBERO_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

```
SELECT /*+ INDEX_DESC(A IDX_01_HINT_TEST3) */ A.SEL
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60'
AND ROWNUM = 1;
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------
   1  COUNT (STOP NODE) (STOP LIMIT 2) (Cost:1272, %%CPU:0, Rows:1)
   2    FILTER (Cost:1272, %%CPU:0, Rows:750)
   3      TABLE ACCESS (ROWID): HINT_TEST3 (Cost:1272, %%CPU:0, Rows:750)
   4        INDEX (SKIP SCAN): IDX_01_TIBERO_TEST3 (Cost:758, %%CPU:0, Rows:750)


Predicate Information
----------------------------------------------------------------------------------------------------
----------------------------------------------------------------
   1   filter: (ROWNUM = 1) (0.010)
   2 - filter: ("A"."AGE" < '60') (1.000)
```

- INDEX_DESC 힌트를 사용하여 SORT 는 없어졌지만 성능의 차이는 없다.
- MAX & INDEX_DESC 사용 모두 IDX_01_TIBERO_TEST3 의 INDEX 가 효과적이지 않아 COST 가 많이 발생한다.

추가 INDEX 생성 후 MIN 함수와 INDEX_ASC HINT 와 비교

```
CREATE INDEX IDX_02_HINT_TEST3 ON HINT_TEST3 (SEL, NAME, AGE);


-- INDEX 추가 생성 후 해당 INDEX로 INDEX_DESC 힌트 사용
SELECT /*+ INDEX_DESC(A IDX_02_HINT_TEST3) */ A.SEL
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60'
AND ROWNUM = 1;
```

```
Execution Plan
----------------------------------------------------------------------------------------
----------------------------------------------------------------
   1  COUNT (STOP NODE) (STOP LIMIT 2) (Cost:7, %%CPU:0, Rows:1)
   2    FILTER (Cost:7, %%CPU:0, Rows:750)
   3      INDEX (FULL) DESCENDING: IDX_02_HINT_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------------------
----------------------------------------------------------------
   1   filter: (ROWNUM = 1) (0.010)
   2   filter: ("A"."NAME" = 'A123') AND ("A"."AGE" < '60') (0.312 * 1.000)


Note
----------------------------------------------------------------------------------------
----------------------------------------------------------------
   3   dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------
----------------------------------------------------------------
   1  COUNT (STOP NODE) (STOP LIMIT 2) (Time:0. ms, Rows:0, Starts:0)
   2    FILTER (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (FULL) DESCENDING: IDX_02_HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

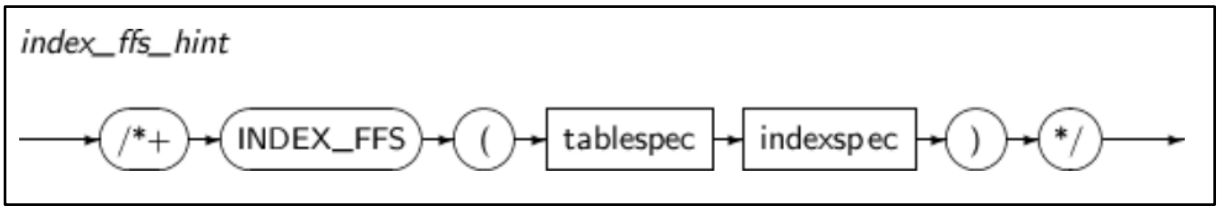
```
-- INDEX 추가 생성 후 해당 INDEX로 MIN 함수 사용
SELECT MAX(A.SEL)
FROM HINT_TEST3 A
WHERE A.NAME = 'A123'
AND A.AGE < '60';
```

```
Execution Plan
----------------------------------------------------------------------------------------
----------------------------------------------------------------
   1  COLUMN PROJECTION (Cost:7, %%CPU:0, Rows:1)
   2    SORT AGGR (Cost:7, %%CPU:0, Rows:1)
   3      COUNT (STOP NODE) (STOP LIMIT 2) (Cost:7, %%CPU:0, Rows:1)
   4        FILTER (Cost:7, %%CPU:0, Rows:750)
   5          INDEX (FULL) DESCENDING: IDX_02_HINT_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------------------
----------------------------------------------------------------
   3 - filter: ("A"."SEL" IS NOT NULL) (1.000)
   4 - filter: ("A"."NAME"   'A123') AND ("A"."AGE" < '60') (0.312 * 1.000)
```

IDX_02_HINT_TEST3 은 출력부 및 조건부에 해당하는 컬럼(SEL, NAME, AGE)이 포함된 결합인덱스로 INDEX 만 읽어서 한번에 FILTER 되어 실행계획이 단순해 졌으며 비용이 많이 소요된 TABLE ACCESS 가 제거 되었다.

INDEX_FFS

- 명시한 테이블에 대해 명시한 인덱스를 사용하여 빠른 전체 인덱스 스캔(Fast Full Index Scan)을 사용하도록 지시하는 힌트이다.
- 인덱스 구조를 무시하고 인덱스 세그먼트 전체를 Multiblock Read 방식으로 스캔한다.
- 문법



시나리오 수행_INDEX_FFS

시나리오 내용
HINT_TEST3 테이블 TRACE
FFS INDEX HINT 사용 TRACE
FULL SCAN INDEX 유도 (성능비교)

시나리오 수행내역
HINT_TEST3 테이블 TRACE

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT

select
 A.name, A.age from HINT_TEST3 A
where A.age < 80;
```

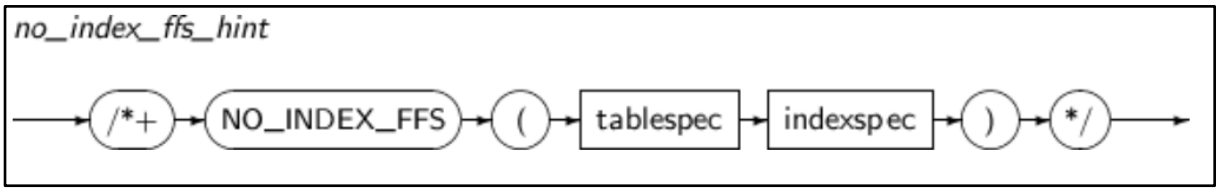
```
Execution Plan
----------------------------------------------------------------------------
----------------------
   1  FILTER (Cost:6, %%CPU:0, Rows:240)
   2    INDEX (FAST FULL SCAN): IDX_02_HINT_TEST3 (Cost:6, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------
----------------------
   1 - filter: ("A"."AGE" < 80) (0.100)


Note
----------------------------------------------------------------------------
----------------------
   2 - dynamic sampling used for this table (32 blocks)

Execution Stat

----------------------
   1  FILTER (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (FAST FULL SCAN): IDX_02_HINT_TEST3 (Time:0. ms, Rows:0, Starts:
0)
```

```
select /*+ INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;
```

```
SQL> select /*+ INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;   2    3

SQL ID: 5hqdjcyh8qykv
Child number: 1130
Plan hash value: 2433113136

Execution Plan
----------------------------------------------------------------------------

   1  FILTER (Cost:6, %%CPU:0, Rows:240)
   2    INDEX (FAST FULL SCAN): IDX_02_HINT_TEST3 (Cost:6, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------
----------------------
   1 - filter: ("A"."AGE" < 80) (0.100)
```

NO_INDEX_FFS

- 명시한 테이블에 대해 명시한 인덱스를 사용하여 빠른 전체 인덱스 스캔(Fast Full Index Scan)을 사용하지않도록 지시하는 힌트이다.
- 문법



시나리오 수행_NO_INDEX_FFS

시나리오 내용
INDEX_FFS 로 수행된다는 가정
NO_INDEX_FFS 로 수행

시나리오 수행내역
INDEX_FFS 로 수행된다는 가정

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


select /*+ INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;
```

```
SQL> select /*+ INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;    2    3

SQL ID: 5hqdjcyh8qykv
Child number: 1130
Plan hash value: 2433113136

Execution Plan
----------------------------------------------------------------------------------------
   1  FILTER (Cost:6, %%CPU:0, Rows:240)
   2    INDEX (FAST FULL SCAN): IDX_02_HINT_TEST3 (Cost:6, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------------------
   1 - filter: ("A"."AGE" < 80) (0.100)


Note

   2   dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------
   1  FILTER (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (FAST FULL SCAN): IDX_02_HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

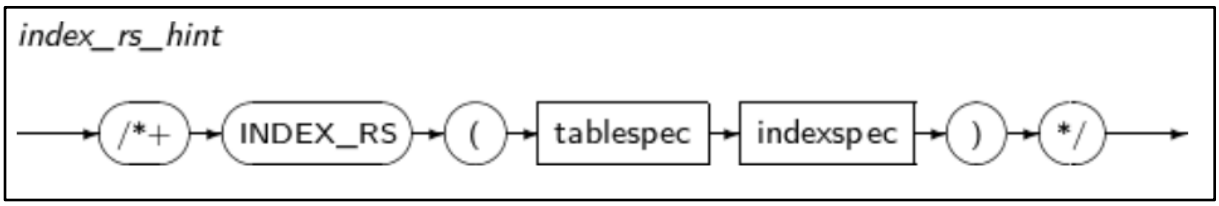
NO_INDEX_FFS 로 수행

```
select /*+ NO_INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;
```

```
SQL> select /*+ NO_INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;   2    3

SQL ID: b7y6rqava25tu
Child number: 1140
Plan hash value: 1257821734

Execution Plan
----------------------------------------------------------------------
   1  FILTER (Cost:7, %%CPU:0, Rows:240)
   2    INDEX (FULL): IDX_02_HINT_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------
   1 - filter: ("A"."AGE" < 80) (0.100)


Note
----------------------------------------------------------------------
   2 - dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------
   1  FILTER (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (FULL): IDX_02_HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

INDEX_RS

- 명시한 테이블에 대해 명시한 인덱스를 사용하여 범위 인덱스 스캔(Range Index Scan)을 사용하도록 지시하는 힌트이다.
- 문법



시나리오 수행_INDEX_RS

시나리오 내용
INDEX RANGE SCAN 사용
TRACE 확인

시나리오 수행내역
INDEX RANGE SCAN 사용 & TRACE 확인

```
DESC HINT_TEST1
DESC HINT_TEST2
```

```
SQL> DESC HINT_TEST1

COLUMN_NAME                                       TYPE
CONSTRAINT
------------------------------------------------- ------------------
--------------------------
NAME                                              VARCHAR(20)
SEX                                               VARCHAR(1)


INDEX_NAME                                TYPE                      COLUMN_
NAME
----------------------------------------- ------------------------- -------
---------------------------
IDX_02_HINT_TEST1                         NORMAL                    NAME
                                                                    SEX

SQL> DESC HINT_TEST2

COLUMN_NAME                                       TYPE
CONSTRAINT
------------------------------------------------- ------------------
--------------------------
NAME                                              VARCHAR(20)
ADDR                                              VARCHAR(50)
PHONE                                             VARCHAR(20)


INDEX_NAME                                TYPE                      COLUMN_
NAME
----------------------------------------- ------------------------- -------
---------------------------
IDX_01_HINT_TEST2                         NORMAL                    NAME
```

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


SELECT /*+ INDEX_RS(A IDX_02_HINT_TEST1) INDEX_RS(B
IDX_01_HINT_TEST02) */
 B.NAME, B.PHONE
FROM HINT_TEST1 A,
 HINT_TEST2 B
WHERE B.NAME BETWEEN 'HUMAN00000771' AND 'HUMAN00000774'
AND A.SEX='1'
AND B.NAME = A.NAME;
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:4, %%CPU:0, Rows:1)
   2    FILTER (Cost:2, %%CPU:0, Rows:1)
   3      INDEX (RANGE SCAN): IDX_02_HINT_TEST1 (Cost:2, %%CPU:0, Rows:1)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   5      FILTER (Cost:2, %%CPU:0, Rows:1)
```

NO_INDEX_FFS 로 수행

```
select /*+ NO_INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;
```

```
SQL> select /*+ NO_INDEX_FFS(A IDX_02_HINT_TEST3) */
 A.name, A.age from HINT_TEST3 A
where A.age < 80;   2    3

SQL ID: b7y6rqava25tu
Child number: 1140
Plan hash value: 1257821734

Execution Plan
----------------------------------------------------------------------------
   1  FILTER (Cost:7, %%CPU:0, Rows:240)
   2    INDEX (FULL): IDX_02_HINT_TEST3 (Cost:7, %%CPU:0, Rows:2400)


Predicate Information
----------------------------------------------------------------------------
   1 - filter: ("A"."AGE" < 80) (0.100)


Note
----------------------------------------------------------------------------
   2 - dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------
   1  FILTER (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (FULL): IDX_02_HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

NO_INDEX_RS

- 명시한 테이블에 대해 명시한 인덱스를 사용하여 범위 인덱스 스캔(Range Index Scan)을 사용하지 않도록 지시하는 힌트이다.
- 문법

no_index_rs_hint

시나리오 수행_NO_INDEX_RS

시나리오 내용
INDEX RANGE SCAN 사용

NO_INDEX RANGE SCAN 사용

시나리오 수행내역

INDEX RANGE SCAN 사용 & TRACE 확인

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


SELECT
 B.NAME, B.PHONE
FROM HINT_TEST2 B
WHERE B.NAME BETWEEN 'HUMAN00000771' AND 'HUMAN00000774';
```

```
SQL> SELECT
 B.NAME, B.PHONE
FROM HINT_TEST2 B
WHERE B.NAME BETWEEN 'HUMAN00000771' AND 'HUMAN00000774';   2    3    4

SQL_ID: 49w1nqcmq075q
Child number: 1155
Plan hash value: 2270201175

Execution Plan

   1  TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   2    INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Cost:2, %%CPU:0, Rows:1)


Predicate Information
----------------------------------------------------------------------------------------------------
   2 - access: ("B"."NAME" >= 'HUMAN00000771') AND ("B"."NAME" <= 'HUMAN00000774') (0.240 * 0.754)


Note
----------------------------------------------------------------------------------------------------
   2   dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------------------
   1  TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   2    INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
```

NO_INDEX RANGE SCAN 사용 & TRACE 확인

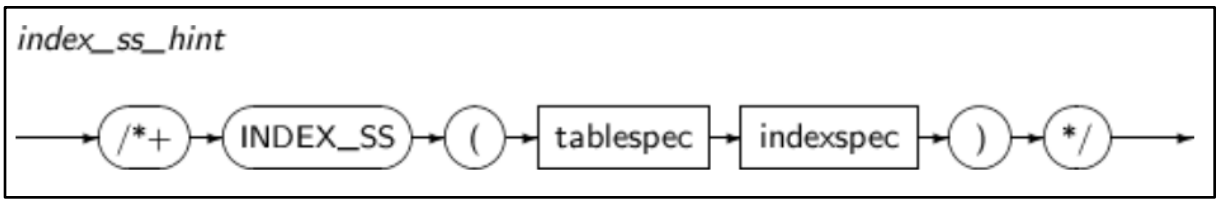
```
SELECT /*+ NO_INDEX_RS(B) */
 B.NAME, B.PHONE
FROM HINT_TEST2 B
WHERE B.NAME BETWEEN 'HUMAN00000771' AND 'HUMAN00000774';
```

```
SQL> SELECT /*+ NO_INDEX_RS(B) */
 B.NAME, B.PHONE
FROM HINT_TEST2 B
WHERE B.NAME BETWEEN 'HUMAN00000771' AND 'HUMAN00000774';   2    3    4

SQL_ID: cybwph2h2zb2u
Child number: 1156
Plan hash value: 4007152058

Execution Plan
----------------------------------------------------------------------------------------------------
   1  TABLE ACCESS (ROWID): HINT_TEST2 (Cost:0, %%CPU:0, Rows:186)
   2    FILTER (Cost:6, %%CPU:0, Rows:186)
   3      INDEX (FAST FULL SCAN): IDX_01_HINT_TEST2 (Cost:6, %%CPU:0, Rows:1000)


Predicate Information
----------------------------------------------------------------------------------------------------
   2 - filter: ("B"."NAME" >= 'HUMAN00000771') AND ("B"."NAME" <= 'HUMAN00000774') (0.248 * 0.754)


Note
-----
   3   dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------------------------------
   1  TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   2    FILTER (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (FAST FULL SCAN): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
```

비용 증가

INDEX_SS

- 명시한 테이블에 대해 명시한 인덱스를 사용하여 인덱스 스킵 스캔(Index Skip Scan)을 사용하도록 지시하는 힌트이다.
- 문법



시나리오 수행_INDEX_SS

시나리오 내용
FULL SCAN 사용

INDEX_SS 힌트 사용

시나리오 수행내역
FULL SCAN 사용

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


SELECT *
FROM HINT_TEST3 A
WHERE NAME = 'A223';
```

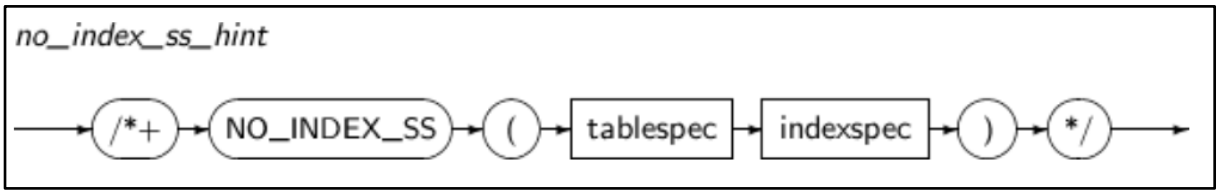
```
SQL> SELECT *
FROM HINT_TEST3 A
WHERE NAME = 'A223';   2    3

SQL ID: 0tqfw61bmusxp
Child number: 1164
Plan hash value: 2600951733

Execution Plan
--------------------------------------------------------------------------------
---------------------------
   1  TABLE ACCESS (FULL): HINT_TEST3 (Cost:24, %%CPU:0, Rows:900)


Predicate Information
--------------------------------------------------------------------------------
---------------------------
   1 - filter: ("A"."NAME" = 'A223') (0.375)


Note
--------------------------------------------------------------------------------
---------------------------
   1 - dynamic sampling used for this table (32 blocks)


Execution Stat
--------------------------------------------------------------------------------
---------------------------
   1  TABLE ACCESS (FULL): HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

```
SELECT /*+ INDEX_SS(A) */ *
FROM HINT_TEST3 A
WHERE NAME = 'A223';
```

```
SQL> SELECT /*+ INDEX_SS(A) */ *
FROM HINT_TEST3 A
WHERE NAME = 'A223';   2    3

SQL ID: 8acbr58u54pcx
Child number: 1165
Plan hash value: 3269331575

Execution Plan
--------------------------------------------------------------------------------
---------------------------
   1  TABLE ACCESS (ROWID): HINT_TEST3 (Cost:1374, %%CPU:0, Rows:900)
   2    INDEX (SKIP SCAN): IDX_01_TIBERO_TEST3 (Cost:758, %%CPU:0, Rows:9
00)
```

NO_INDEX_SS

- 명시한 테이블에 대해 명시한 인덱스를 사용하여 인덱스 스킵 스캔(Index Skip Scan)을 사용하지않도록 지시하는 힌트이다.
- 문법



시나리오 수행_NO_INDEX_SS

시나리오 내용
NO_INDEX_SS

시나리오 수행내역
NO_INDEX_sS

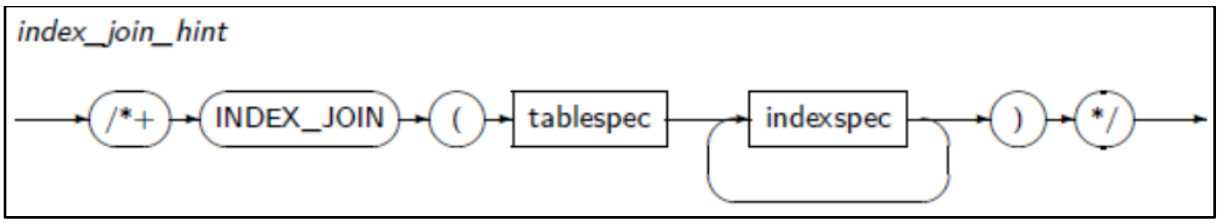
```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


SELECT /*+ NO_INDEX_SS(A) */
 *
FROM HINT_TEST3 A
WHERE NAME = 'A223';
```

```
SQL> SELECT /*+ NO_INDEX_SS(A) */
 *
FROM HINT_TEST3 A
WHERE NAME = 'A223';   2    3    4

SQL ID: 4hf8frc6385zq
Child number: 1211
Plan hash value: 2600951733

Execution Plan
----------------------------------------------------------------------------
---------------------------
   1  TABLE ACCESS (FULL): HINT_TEST3 (Cost:24, %%CPU:0, Rows:900)


Predicate Information
----------------------------------------------------------------------------
---------------------------
   1 - filter: ("A"."NAME" = 'A223') (0.375)


Note
----------------------------------------------------------------------------
---------------------------
   1 - dynamic sampling used for this table (32 blocks)


Execution Stat
----------------------------------------------------------------------------
---------------------------
   1  TABLE ACCESS (FULL): HINT_TEST3 (Time:0. ms, Rows:0, Starts:0)
```

결과 : 더이상 줄일 수 있는 모수가 없기 때문에 INDEX 보다 FULL SCAN 으로 읽는것이 더 유리하다.

INDEX_JOIN

- 명시한 테이블에 대해 명시한 두 개 이상의 힌트를 사용하여, 테이블을 스캔할 때 자체 조인(Self Join)을 사용하도록 지시하는 힌트이다.
- 문법

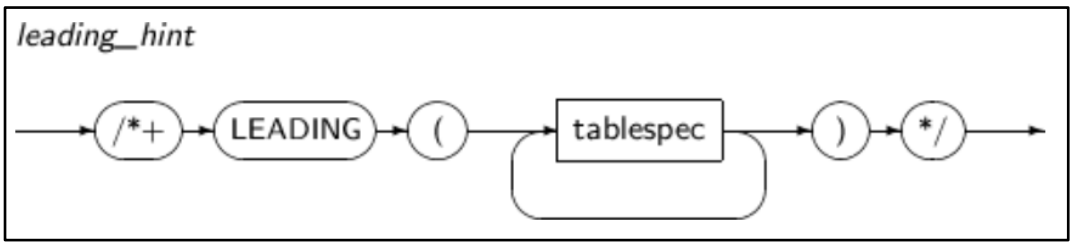


2.8.4. 조인 순서

- LEADING ORDERED 보다 더 많이 사용된다.

LEADING

- 조인에서 먼저 조인되어야 할 테이블의 집합을 명시하는 힌트이다.
- 만일 ORDERED 힌트가 사용되는 경우에는 LEADING 힌트는 모두 무시된다.
- 문법



시나리오 수행_LEADING

시나리오 내용
원본
LEADING 사용

시나리오 수행내역
원본

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


--A,B 순서로 읽음
SELECT A.SEX, A.NAME, B.PHONE
FROM HINT_TEST1 A,
 HINT_TEST2 B
WHERE A.SEX = '1'
AND A.NAME = B.NAME
AND B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774');
```

```
SQL> --A,B 순서로 읽음
SELECT A.SEX, A.NAME, B.PHONE
FROM HINT_TEST1 A,
 HINT_TEST2 B
WHERE A.SEX = '1'
AND A.NAME = B.NAME
AND B.NAME IN ('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774');
SQL>    2    3    4    5    6
SQL ID: ar5g7af793597
Child number: 1172
Plan hash value: 2646062339

Execution Plan

   1  INDEX JOIN (Cost:7, %%CPU:0, Rows:2)
   2    FILTER (Cost:5, %%CPU:0, Rows:2)
   3      INDEX (FULL): IDX_02_HINT_TEST1 (Cost:5, %%CPU:0, Rows:1000)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   5      FILTER (Cost:2, %%CPU:0, Rows:1)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Cost:2, %%CPU:0, Rows:1)


Predicate Information

   2 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) AND ("A"."SEX" = '1') (0.005 * 0.499)

   5   filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   6 - access: ("B"."NAME" = "A"."NAME") (0.200)
```

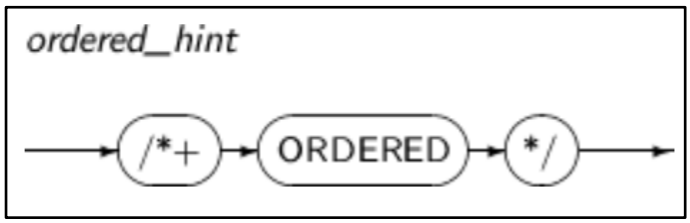
LEADING 사용

```
--B,A 순서로 읽음
SELECT /*+ LEADING(B A) */
 A.SEX, A.NAME, B.PHONE
FROM HINT_TEST1 A,
 HINT_TEST2 B
WHERE A.SEX = '1'
AND A.NAME = B.NAME
AND B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774');
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:9, %%CPU:0, Rows:2)
   2    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:6, %%CPU:0, Rows:5)
   3      FILTER (Cost:5, %%CPU:0, Rows:5)
   4        INDEX (FULL): IDX_01_HINT_TEST2 (Cost:5, %%CPU:0, Rows:1000)
   5    FILTER (Cost:2, %%CPU:0, Rows:1)
   6      INDEX (RANGE SCAN): IDX_02_HINT_TEST1 (Cost:2, %%CPU:0, Rows:1)


Predicate Information
----------------------------------------------------------------------------------------------------
   3 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   5 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.005)

   6   access: ("B"."NAME" = "A"."NAME") AND ("A"."SEX" = '1') (0.200 * 0.499)


Note

   4   dynamic sampling used for this table (32 blocks)


Execution Stat

   1  INDEX JOIN (Time:0. ms, Rows:0, Starts:0)
   2    TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   3      FILTER (Time:0. ms, Rows:0, Starts:0)
   4        INDEX (FULL): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   5    FILTER (Time:0. ms, Rows:0, Starts:0)
   6      INDEX (RANGE SCAN): IDX_02_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
```

ORDERED

- 테이블을 FROM 절에 명시된 순서대로 조인하도록 지시하는 힌트이다.
- 질의 최적화기의 조인 순서를 명확히 알고 있을 경우에만 ORDERED 힌트를 사용하는 것이 좋다.
- 문법



시나리오 수행_ORDERED

시나리오 내용
원본
ORDERED 사용

시나리오 수행내역
원본

```
SET LINES 200
SET AUTOT TRACEONLY EXP PLANSTAT


SELECT A.SEX, A.NAME, B.PHONE
FROM HINT_TEST2 B,
 HINT_TEST1 A
WHERE A.SEX = '1'
AND A.NAME = B.NAME
AND B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774');
```

```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:7, %%CPU:0, Rows:2)
   2    FILTER (Cost:5, %%CPU:0, Rows:2)
   3      INDEX (FULL): IDX_02_HINT_TEST1 (Cost:5, %%CPU:0, Rows:1000)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:3, %%CPU:0, Rows:1)
   5      FILTER (Cost:2, %%CPU:0, Rows:1)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Cost:2, %%CPU:0, Rows:1)


Predicate Information
----------------------------------------------------------------------------------------------------
   2 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) AND ("A"."SEX" = '1') (0.004 * 0.499)

   5   filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.004)

   6   access: ("A"."NAME" = "B"."NAME") (0.200)


Note
----------------------------------------------------------------------------------------------------
   6 - dynamic sampling used for this table (32 blocks)


Execution Stat

   1  INDEX JOIN (Time:0. ms, Rows:0, Starts:0)
   2    FILTER (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (FULL): IDX_02_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
   4    TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   5      FILTER (Time:0. ms, Rows:0, Starts:0)
   6        INDEX (RANGE SCAN): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
```

ORDERED 사용

```
SELECT /*+ ORDERED */
 A.SEX, A.NAME, B.PHONE
FROM HINT_TEST2 B,
 HINT_TEST1 A
WHERE A.SEX = '1'
AND A.NAME = B.NAME
AND B.NAME IN
('HUMAN00000771','HUMAN00000772','HUMAN00000773','HUMAN00000774');
```

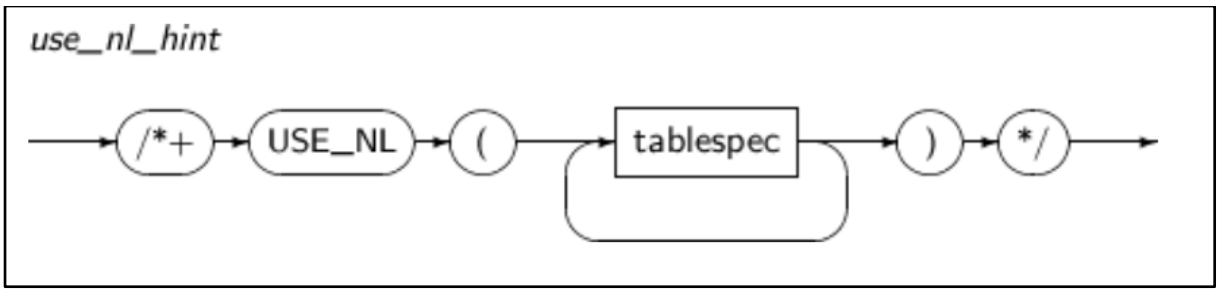
```
Execution Plan
----------------------------------------------------------------------------------------------------
   1  INDEX JOIN (Cost:9, %%CPU:0, Rows:2)
   2    TABLE ACCESS (ROWID): HINT_TEST2 (Cost:6, %%CPU:0, Rows:5)
   3      FILTER (Cost:5, %%CPU:0, Rows:5)
   4        INDEX (FULL): IDX_01_HINT_TEST2 (Cost:5, %%CPU:0, Rows:1000)
   5    FILTER (Cost:2, %%CPU:0, Rows:1)
   6      INDEX (RANGE SCAN): IDX_02_HINT_TEST1 (Cost:2, %%CPU:0, Rows:1)


Predicate Information

   3 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.004)

   5 - filter: (("B"."NAME") IN (('HUMAN00000771'),('HUMAN00000772'),('HUMAN00000773'),('HUMAN000007
74'))) (0.004)

   6 - access: ("A"."NAME" = "B"."NAME") AND ("A"."SEX" = '1') (0.200 * 0.499)


Note

   4   dynamic sampling used for this table (32 blocks)


Execution Stat

   1  INDEX JOIN (Time:0. ms, Rows:0, Starts:0)
   2    TABLE ACCESS (ROWID): HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   3      FILTER (Time:0. ms, Rows:0, Starts:0)
   4        INDEX (FULL): IDX_01_HINT_TEST2 (Time:0. ms, Rows:0, Starts:0)
   5    FILTER (Time:0. ms, Rows:0, Starts:0)
   6      INDEX (RANGE SCAN): IDX_02_HINT_TEST1 (Time:0. ms, Rows:0, Starts:0)
```

2.8.5. 조인 방법

- 조인 방법이 적용된 힌트는 한 테이블에 대해서만 조인 방법을 지시한다.

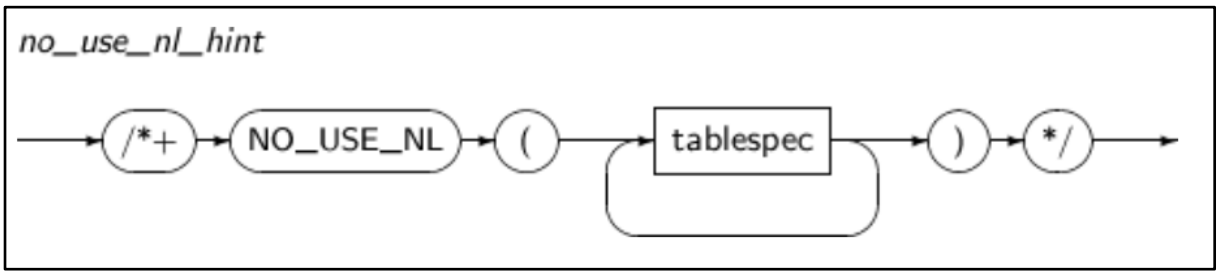
USE_NL

- 명시한 테이블을 다른 테이블과 조인하는 경우 중첩 루프 조인을 사용하도록 지시하는 힌트이다.
- 문법



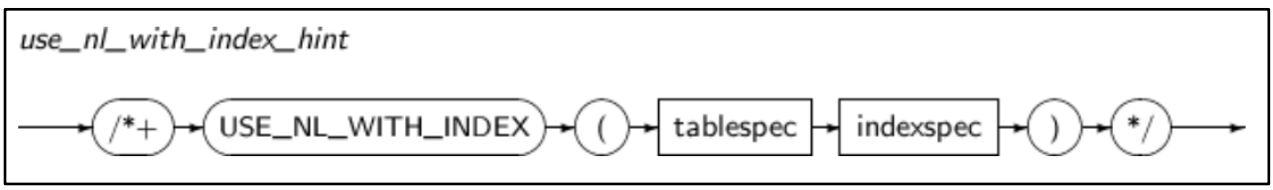
NO_USE_NL

- 명시한 테이블을 다른 테이블과 조인하는 경우 중첩 루프 조인을 사용하지 않도록 지시하는 힌트이다.
- 문법



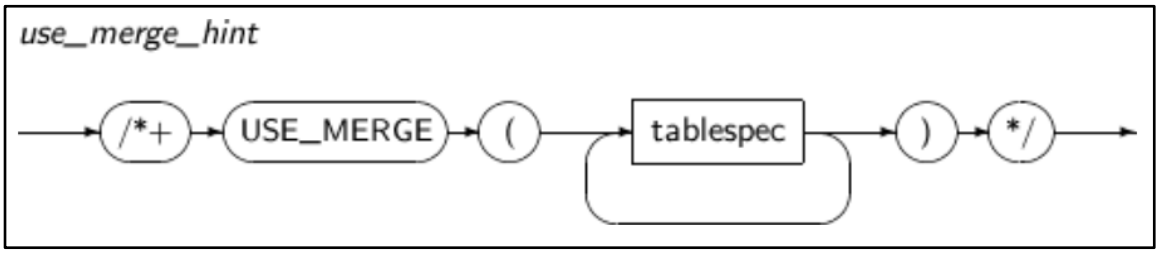
USE_NL_WITH_INDEX

- 명시한 테이블을 다른 테이블과 조인하는 경우 중첩 루프 조인을 사용하도록 지시하는 힌트이다.
- 명시한 테이블에 대한 접근은 명시한 인덱스와 두 테이블에 대한 조인 조건을 이용하여 이루어져야 한다.
- 문법



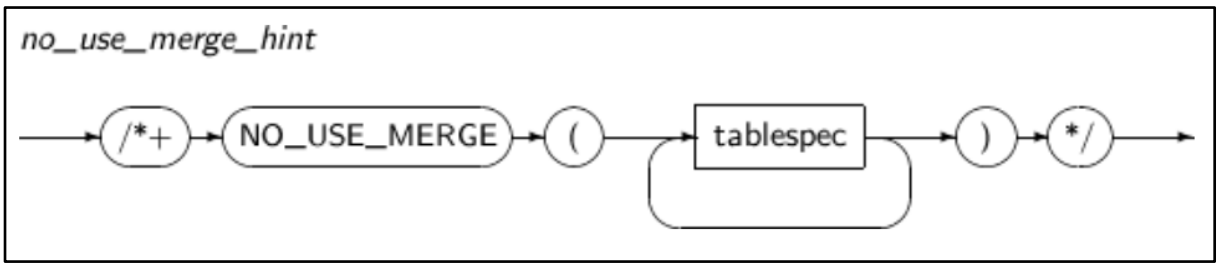
USE_MERGE

- 명시한 테이블을 다른 테이블과 조인하는 경우 합병 조인을 사용하도록 지시하는 힌트이다.
- 문법



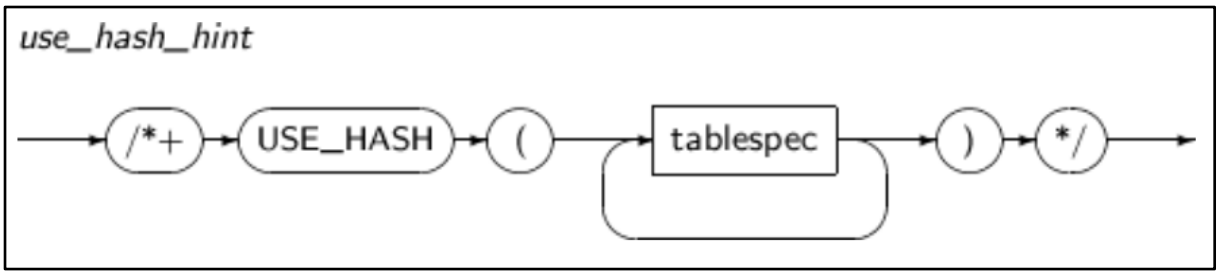
NO_USE_MERGE

- 명시한 테이블을 다른 테이블과 조인하는 경우 합병 조인을 사용하지 않도록 지시하는 힌트이다.
- 문법



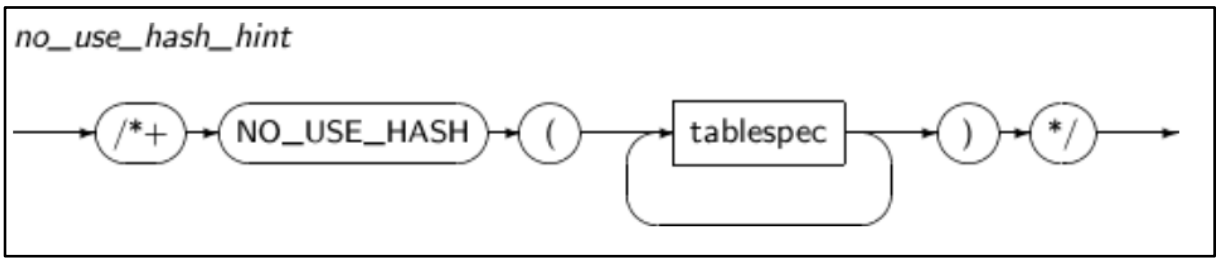
USE_HASH

- 명시한 테이블을 다른 테이블과 조인하는 경우 해시 조인을 사용하도록 지시하는 힌트이다.
- 문법



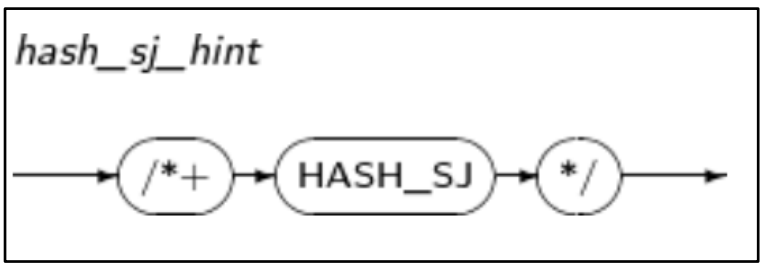
NO_USE_HASH

- 명시한 테이블을 다른 테이블과 조인하는 경우 해시 조인을 사용하지 않도록 지시하는 힌트이다.
- 문법



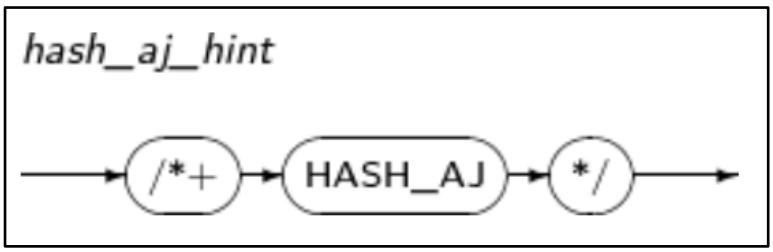
HASH_SJ

- 부질의를 언네스팅할 때 해시방법을 이용한 세미조인으로 하도록 지시하는 힌트이다.
- 문법



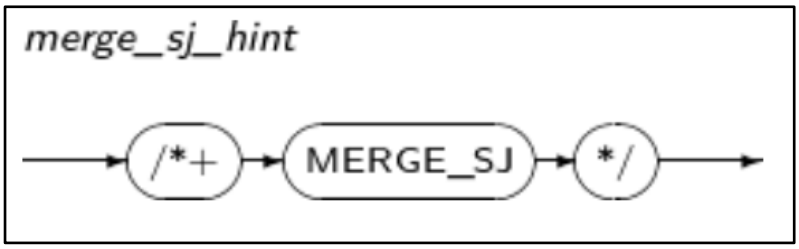
HASH_AJ

- 부질의를 언네스팅할 때 해시방법을 이용한 안티조인으로 하도록 지시하는 힌트이다.
- 문법



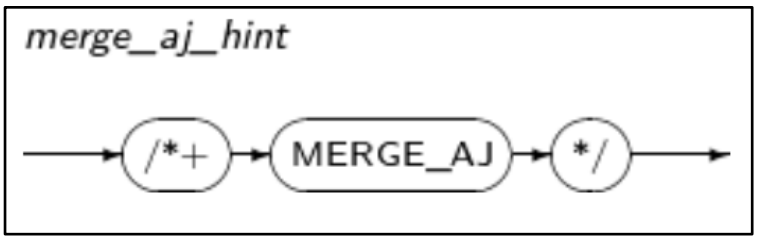
MERGE_SJ

- 부질의를 언네스팅할 때 머지방법을 이용한 세미조인으로 하도록 지시하는 힌트이다.
- 문법



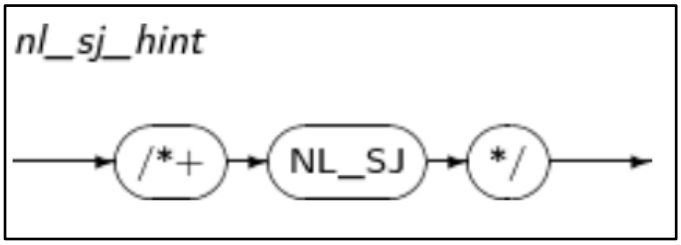
MERGE_AJ

- 부질의를 언네스팅할 때 머지방법을 이용한 안티조인으로 하도록 지시하는 힌트이다.
- 문법



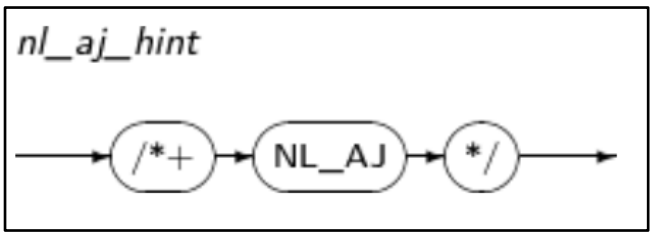
NL_SJ

- 부질의를 언네스팅할 때 네스티드 루프 방법을 이용한 세미조인으로 하도록 지시하는 힌트이다.
- 문법



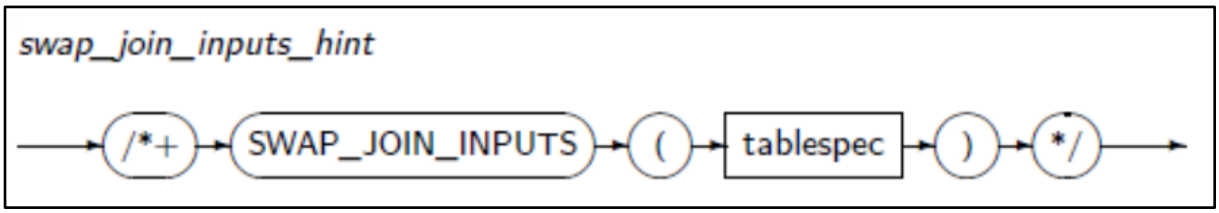
NL_AJ

- 부질의를 언네스팅할 때 네스티드 루프 방법을 이용한 안티조인으로 하도록 지시하는 힌트이다.
- 문법



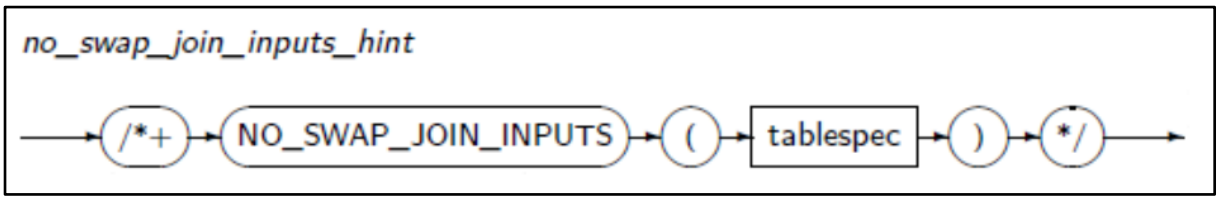
SWAP_JOIN_INPUTS

- 해시 조인을 수행하는 경우 명시한 테이블을 사용하여 해시 테이블을 빌드하도록 지시하는 힌트이다.
- 문법



NO_SWAP_JOIN_INPUTS

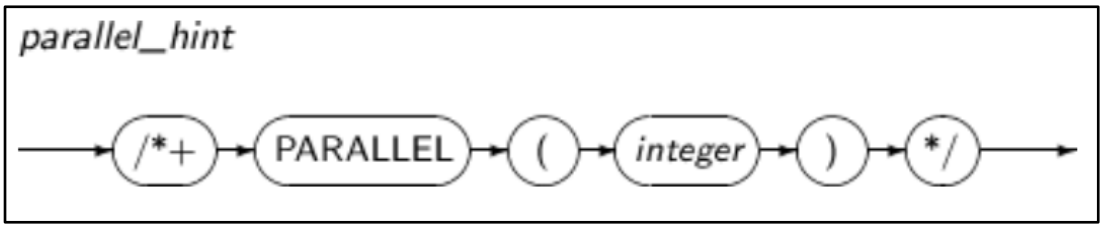
- 해시 조인을 수행하는 경우 명시한 테이블을 사용하여 해시 테이블을 빌드되지 않도록 지시하는 힌트이다.
- 문법



2.8.6. 병렬 처리

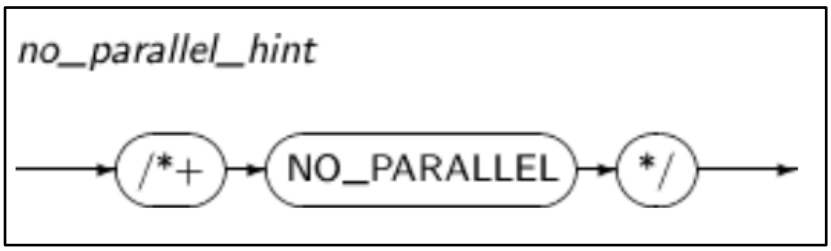
PARALLEL

- 지정한 개수의 스레드를 사용해 질의의 수행을 병렬로 진행하도록 지시하는 힌트이다.
- 문법



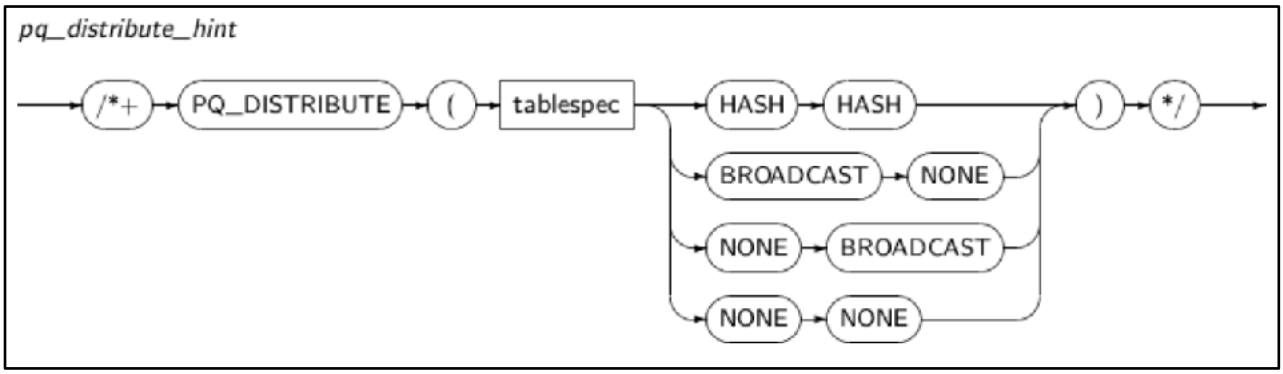
NO_PARALLEL

- 지정한 개수의 스레드를 사용해 질의의 수행을 병렬로 진행하지 않도록 지시하는 힌트이다.
- 문법



PQ_DISTRIBUTE

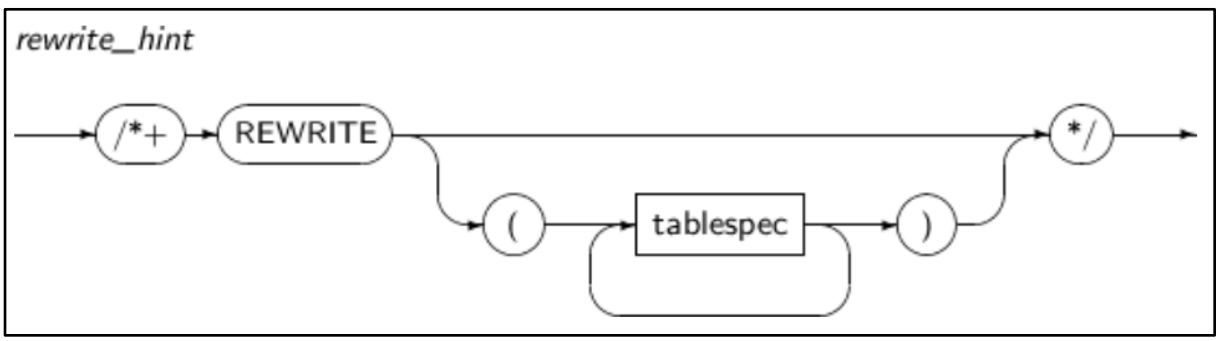
- 조인을 포함한 질의의 병렬 처리에서 조인될 로우의 분산 방법을 지시하는 힌트이다.
- 문법



2.8.7. 실체화 뷰

REWRITE

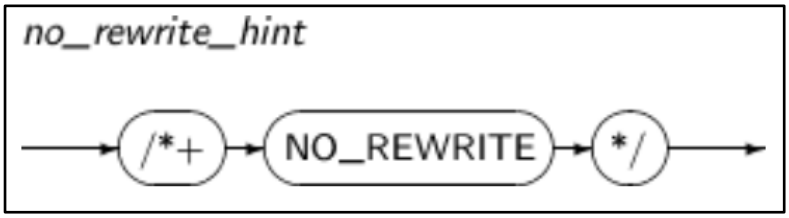
- 해당 질의 블록에서 비용의 비교 없이 실체화 뷰를 사용하여 질의의 다시 쓰기를 하도록 지시하는 힌트이다.
- REWRITE 힌트가 사용된 질의 블록만 다시 쓰기를 한 결과와 모든 블록에서 다시 쓰기를 한 결과의 비용을 비교해서 더 좋은 쪽을 질의 최적화기가 선택하게 된다.
- 문법



NO_REWRITE

- 해당 질의 블록에서는 질의의 다시 쓰기를 하지 않도록 지시하는 힌트이다.

- 문법



2.9. 스키마 객체

2.9.1. 테이블

- 데이터를 저장하기 위한 가장 기본적인 저장 단위이다.
- 이 테이블은 행과 열로 구성된 2 차원 행렬의 형태를 갖는다.

2.9.2. 인덱스

- 테이블과 별도의 저장공간을 이용하여 그 테이블의 특정 컬럼에 대하여 빠른 검색을 가능하게 하는 데이터 구조이다.
    - 자동 인덱싱
    - 칼럼의 중복 허용
    - 복수 칼럼 허용
    - 인덱스의 적용
    - 인덱스의 관리
    - 인덱스의 제거

2.9.3. 뷰

- SQL 질의 문장에 대하여 이름을 붙인 것으로, 빈번히 수행되는 질의의 결과를 테이블 형태로 이용할 수 있도록 정의한다.
- 하나의 테이블을 여러 사용자가 함께 접근할 수 있지만 사용자에 따라 테이블 내용의 일부를 숨김으로써 정보 보안을 유지하고자 한다.
- 실제 공간을 차지하지 않다 (view 가 data 를 가지고 있지않다.)

2.9.4. 시퀀스

- 티베로 데이터베이스에서 유일한 연속적인 값을 생성해 내는 객체이다.
    - CURRVAL : 현재 세션에서 마지막으로 조회한 NEXTVAL 값을 반환
    - NEXTVAL : 시퀀스의 현재 값을 증가시키고 증가된 그 값을 반환

2.9.5. 동의어

- 특정 객체에 대하여 정의하는 ALIAS 와 같다.

시나리오 수행_스키마 객체

시나리오 내용
TEST 접속
테이블 생성
인덱스 생성
뷰 생성
시퀀스 생성
동의어 생성

시나리오 수행내역
TEST 사용자 접속
CONN TEST/TEST
테이블 생성

```
CREATE TABLE DEPARTMENT
(DEPTNO NUMBER(2) CONSTRAINT DEP_DEPTNO_NN NOT NULL,
DNAME VARCHAR2(14),
LOC VARCHAR2(13),
CONSTRAINT DEP_ID_PK PRIMARY KEY(DEPTNO));
```

```
SQL> CREATE TABLE DEPARTMENT
(DEPTNO NUMBER(2) CONSTRAINT DEP_DEPTNO_NN NOT NULL,
DNAME VARCHAR2(14),
LOC VARCHAR2(13),
CONSTRAINT DEP_ID_PK PRIMARY KEY(DEPTNO));   2    3    4    5

Table 'DEPARTMENT' created.
```

```
CREATE TABLE EMPLOYEE
(EMPNO NUMBER(4) CONSTRAINT EMP_EMPNO_NN NOT NULL,
ENAME VARCHAR2(15) CONSTRAINT EMP_ENAME_NN NOT NULL,
JOB VARCHAR2(15), MANAGERNO NUMBER(4),
STARTDATE DATE,
SALARY NUMBER(10),
DEPTNO NUMBER(3),
CONSTRAINT EMP_ID_PK PRIMARY KEY(EMPNO),
CONSTRAINT EMP_MGR_FK FOREIGN KEY(MANAGERNO) REFERENCES
EMPLOYEE(EMPNO),
CONSTRAINT EMP_DEPTNO_FK FOREIGN KEY(DEPTNO) REFERENCES
DEPARTMENT(DEPTNO));
```

```
SQL> CREATE TABLE EMPLOYEE
(EMPNO NUMBER(4) CONSTRAINT EMP_EMPNO_NN NOT NULL,
ENAME VARCHAR2(15) CONSTRAINT EMP_ENAME_NN NOT NULL,
JOB VARCHAR2(15),
MANAGERNO  NUMBER(4),
STARTDATE  DATE,
SALARY  NUMBER(10),
DEPTNO  NUMBER(3),
CONSTRAINT EMP_ID_PK PRIMARY KEY(EMPNO),
CONSTRAINT EMP_MGR_FK FOREIGN KEY(MANAGERNO) REFERENCES EMPLOYEE(EMPNO),
CONSTRAINT EMP_DEPTNO_FK FOREIGN KEY(DEPTNO) REFERENCES DEPARTMENT(DEPTNO));
```

```
DESC DEPARTMENT
DESC EMPLOYEE
```

```
SQL> DESC DEPARTMENT
COLUMN NAME                                       TYPE            CONSTRAINT
------------------------------------------------- --------------- ----------------------------
DEPTNO                                            NUMBER(2)       PRIMARY KEY
                                                                  NOT NULL
DNAME                                             VARCHAR(14)
LOC                                               VARCHAR(13)


INDEX_NAME                          TYPE                    COLUMN_NAME
----------------------------------- ----------------------- --------------------------------
```

결론 : EMPLOYEE 테이블의 외래키로 참조되는 DEPTNO 는 DEPARTMENT 의 PRIMARY KEY 인 칼럼이므로 DEPARTMENT TABLE 생성이 되어있어야 EMPLOYEE 테이블 생성 가능

인덱스 생성

```
CREATE INDEX DEP_NAME_IDX ON DEPARTMENT(DNAME);
DESC DEPARTMENT
```

```
SQL> CREATE INDEX DEP_NAME_IDX ON DEPARTMENT(DNAME);

Index 'DEP_NAME_IDX' created.

SQL> DESC DEPARTMENT

COLUMN_NAME                                       TYPE              CONSTRAINT



DEPTNO                                            NUMBER(2)         PRIMARY KEY
                                                                    NOT NULL
DNAME                                             VARCHAR(14)
LOC                                               VARCHAR(13)


INDEX_NAME                          TYPE                    COLUMN_NAME

----------------------------------- ----------------------- -----------------------
------------
DEP_ID_PK                           NORMAL                  DEPTNO
DEP_NAME_IDX                        NORMAL                  DNAME
```

뷰 생성

```
CONN SYS/TIBERO
GRANT DBA TO TEST;  -VIEW 생성할 수 있는 권한 부여
CONN TEST/TEST


CREATE VIEW EMPLOYEE_VIEW
AS SELECT * FROM EMPLOYEE
WHERE EMPNO LIKE '79%';
SELECT * FROM EMPLOYEE_VIEW;
```

```
SQL> CREATE VIEW EMPLOYEE_VIEW
  2 AS SELECT * FROM EMPLOYEE
  3 WHERE EMPNO LIKE '79%';

View 'EMPLOYEE_VIEW' created.

SQL> SELECT * FROM EMPLOYEE_VIEW
  2 ;

     EMPNO ENAME           JOB              MANAGERNO STARTDATE      SALARY     DEPTNO
---------- --------------- --------------- ---------- ---------- ---------- ----------
      7900 JAMES           CLERK                 7698 0081/12/11        950         30
      7902 FORD            ANALYST               7566 0081/03/12       3000         20
      7990 NEW             ANALYST               7902 0081/05/12       2500         30

3 rows selected.
```

시퀀스 생성

```
DESC USER_SEQUENCES  - 전체 시퀀스 칼럼 조회
CREATE SEQUENCE DEP_SEQ  - 시퀀스 생성
INCREMENT BY 1
START WITH 1
MINVALUE 1
MAXVALUE 10
NOCYCLE
NOCACHE;
```

```
SQL> CREATE SEQUENCE DEP_SEQ
  2 INCREMENT BY 1
  3 START WITH 1
  4 MINVALUE 1
  5 MAXVALUE 10
  6 NOCYCLE
  7 NOCACHE;

Sequence 'DEP_SEQ' created.
```

```
INSERT INTO DEPARTMENT VALUES(DEP_SEQ.NEXTVAL, 'HR','SEOUL');
INSERT INTO DEPARTMENT
VALUES(DEP_SEQ.NEXTVAL,'FINANCE','MILPITAS');
SELECT * FROM DEPARTMENT;

/*ERROR : DEPTNO는 PRIMARY KEY이므로 UNIQUE해야함*/
INSERT INTO DEPARTMENT
VALUES(DEP_SEQ.CURRVAL,'SALESMAN','BUNDANG');
```

```
SQL> INSERT INTO DEPARTMENT VALUES(DEP_SEQ.NEXTVAL, 'HR','SEOUL');

1 row inserted.

SQL> INSERT INTO DEPARTMENT VALUES(DEP_SEQ.NEXTVAL,'FINANCE','MILPITAS');

1 row inserted.

SQL> SELECT * FROM DEPARTMENT;

    DEPTNO DNAME          LOC
---------- -------------- -------------
         1 HR             SEOUL
         2 FINANCE        MILPITAS
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

6 rows selected.
```

동의어 생성

```
CREATE PUBLIC SYNONYM PUB_EMP FOR EMPLOYEE;
SELECT * FROM PUB_EMP;
```

```
SQL> SELECT * FROM PUB_EMP;

     EMPNO ENAME      JOB              MANAGERNO STARTDATE      SALARY     DEPTNO
---------- ---------- --------------- ---------- ---------- ---------- ----------
      7369 SMITH      CLERK                7902 0080/12/09        800         20
      7499 ALLEN      SALESMAN             7839 0081/09/10       1600         30
      7521 WARD       SALESMAN             7698 0081/02/23       1250         30
      7566 JONES      MANAGER              7839 0081/02/04       2975         20
      7654 MARTIN     SALESMAN             7698 0081/02/11       1250         30
      7698 BLAKE      MANAGER              7839 0081/05/01       2850         30
      7782 CLARK      MANAGER              7839 0081/05/09       2450         10
      7788 SCOTT      ANALTST              7566 0082/12/22       3000         20
      7839 KING       PRESIDENT                 0081/11/17       5000         10
      7844 TURNER     SALESMAN             7698 0081/08/21       1500         30
      7876 ADAMS      CLERK                7788 0083/01/15       1100         20
      7900 JAMES      CLERK                7698 0081/12/11        950         30
      7902 FORD       ANALYST              7566 0081/03/12       3000         20
      7990 NEW        ANALYST              7902 0081/05/12       2500         30

14 rows selected.
```

```
SELECT * FROM EMPLOYEE;
/*EMPLOYEE 테이블과 PUB_EMP 는 같은 값이 조회됨*/
```

```
SQL> SELECT * FROM EMPLOYEE;

     EMPNO ENAME      JOB              MANAGERNO STARTDATE      SALARY     DEPTNO
---------- ---------- --------------- ---------- ---------- ---------- ----------
      7369 SMITH      CLERK                7902 0080/12/09        800         20
      7499 ALLEN      SALESMAN             7839 0081/09/10       1600         30
      7521 WARD       SALESMAN             7698 0081/02/23       1250         30
      7566 JONES      MANAGER              7839 0081/02/04       2975         20
      7654 MARTIN     SALESMAN             7698 0081/02/11       1250         30
      7698 BLAKE      MANAGER              7839 0081/05/01       2850         30
      7782 CLARK      MANAGER              7839 0081/05/09       2450         10
      7788 SCOTT      ANALTST              7566 0082/12/22       3000         20
      7839 KING       PRESIDENT                 0081/11/17       5000         10
      7844 TURNER     SALESMAN             7698 0081/08/21       1500         30
      7876 ADAMS      CLERK                7788 0083/01/15       1100         20
      7900 JAMES      CLERK                7698 0081/12/11        950         30
      7902 FORD       ANALYST              7566 0081/03/12       3000         20
      7990 NEW        ANALYST              7902 0081/05/12       2500         30

14 rows selected.
```