2. Data Dictionary, Constraints, View, Sequence, Index

Data Dictionary

- 개요
  - 데이터베이스에 대한 메타정보를 저장하는 테이블 및 뷰(view)들의 집합
  - SYSTEM 사용자 소유
  - Base Tables
    - 문서화가 잘 되어있지 않고 컬럼 이름이 직관적이지 않아 사용이 어려움
  - Data Dictionary Views
    - Base table을 기반으로 정의된 view
    - **Static views** – 데이터베이스에 관한 정보 (예: USER_TABLES)
    - Dynamic views – Oracle instance에 관한 정보 (예: V$PARAMETERS)
    - Global dynamic views – 모든 Oracle instance들에 관한 정보 (예: GV$PARAMETERS)

Data Dictionary

- Static Views
  - 데이터베이스에 저장된 모든 객체, 사용자, 권한, 역할, 저장구조 및 공간 등에 대한 정보를 포함
    - Schema Objects: Tables, Indexes, Constraints, Views, Sequences, Stored Procedures, Triggers, Synonyms, DB Links, …
    - DDL을 통해 생성 및 변경됨
  - 종류
    - 접두사(prefix)를 통해 구분

접두사	범위
USER_	특정 사용자가 생성 및 소유한 객체에 대한 정보
ALL_	특정 사용자가 접근 가능한 객체에 대한 정보
DBA_	데이터베이스 관리자가 접근 가능한 객체(즉, 모든 객체)에 대한 정보

Data Dictionary

- 주요 Static Views
  - USER_TABLES, ALL_TABLES, DBA_TABLES
    - 테이블에 관한 정보
  - USER_CONSTRAINTS, USER_CONS_COLUMNS
    - 제약조건에 관한 정보
  - USER_INDEXES, USER_IND_COLUMNS
    - Index에 관한 정보
  - USER_VIEWS
    - View에 관한 정보
  - USER_SOURCE
    - 저장 프로시저(stored procedure)에 관한 정보
  - DBA_USERS
    - 데이터베이스 사용자에 관한 정보

Data Dictionary

- 사용 예
  - 데이터베이스의 모든 사용자들에 관한 정보 조회

```
SELECT username, password, default_tablespace, account_status
FROM DBA_USERS;
```

  - USERS 테이블스페이스에 저장된 테이블 및 그 소유자에 관한 정보 조회

```
SELECT owner, table_name
FROM DBA_TABLES
WHERE tablespace_name = 'USERS';
```

  - 특정 사용자가 자신이 소유한 DEPT 테이블에 관한 상세정보 조회

```
SELECT tablespace_name, num_rows, avg_row_len, partitioned, nested
FROM USER_TABLES
WHERE table_name = 'DEPT';
// 주의: dictionary에는 모든 객체 이름들이 대문자로 저장됨
```


Data Dictionary

- USER_TABLES 구조

컬럼명	데이터 타입	의미
TABLE_NAME	VARCHAR2(30)	테이블 이름
TABLESPACE_NAME	VARCHAR2(30)	테이블이 속한 테이블스페이스 이름
CLUSTER_NAME	VARCHAR2(30)	테이블이 속한 클러스터 이름
PCT_FREE	NUMBER	Block 내의 최소 free space 크기(%)
PCT_USED	NUMBER	Block 내의 최소 used space 크기(%)
MIN_EXTENT	NUMBER	Segment에 저장되는 최소 extent 개수
MAX_EXTENT	NUMBER	Segment에 저장되는 최대 extent 개수
NUM_ROWS	NUMBER	테이블 내에 저장된 행의 개수
AVG_ROW_LEN	NUMBER	행들의 평균 길이
PARTITIONED	VARCHAR2(3)	테이블이 분할되었는지 여부
NESTED	VARCHAR2(3)	중첩 테이블인지 여부
LAST_ANALYZED	DATE	테이블이 마지막으로 분석된 시각
…		

Integrity Constraints

- 무결성 제약조건
  - 데이터베이스 내의 데이터들의 정확성을 유지
  - 부적절한 데이터가 입력, 저장되는 것을 방지하기 위함

무결성 제약조건	의미
NOT NULL	해당 컬럼 값으로 NULL을 허용하지 않음
Unique	테이블 내에서 해당 컬럼 값은 항상 유일해야 함
Primary Key	해당 컬럼 값은 반드시 존재해야 하고 유일해야 함 (NOT NULL 조건과 Unique 조건을 결합한 형태)
Foreign Key	해당 컬럼 값은 참조되는 테이블의 컬럼의 값을 참조해야 함 (참조되는 컬럼에 없는 값은 저장 불가)
Check	해당 컬럼에 저장 가능한 데이터 값의 범위나 조건을 지정

Integrity Constraints

- 정의 방법
  1. Column-level definition: 각 컬럼에 대해 정의
     - column_name data_type [CONSTRAINT *constraint_name*] constraint_type
     - 예: CREATE TABLE dept (생략 가능)

```
(deptno   NUMBER(2) CONSTRAINT dept_pk PRIMARY KEY,
 dname    VARCHAR2(15),
 loc      VARCHAR2(15));
```

  2. Table-level definition: 컬럼들의 정의가 모두 끝난 후 별도로 정의
     - [CONSTRAINT *constraint_name*] constraint_type (*column_name*)
     - 예: CREATE TABLE dept

```
(deptno   NUMBER(2),
 dname    VARCHAR2(15),
 loc      VARCHAR2(15),
 CONSTRAINT dept_pk PRIMARY KEY (deptno));
```

Integrity Constraints 관련 정보

- Data Dictionary View
  - USER_CONSTRAINT

컬럼명	데이터 타입	의미
OWNER	VARCHAR2(30)	제약조건의 소유자
CONSTRAINT_NAME	VARCHAR2(30)	제약조건 이름
CONSTRAINT_TYPE	VARCHAR2(1)	제약조건의 종류 (P/R/U/C)
TABLE_NAME	VARCHAR2(30)	제약조건이 적용되는 테이블 이름
SEARCH_CONDITION	LONG	Check / NOT NULL 제약조건에 대한 텍스트
R_OWNER	VARCHAR2(30)	참조 테이블의 소유자 (FK 제약조건에서 사용됨)
R_CONSTRAINT_NAME	VARCHAR2(30)	참조 테이블의 Unique/PK 제약조건 이름 (FK 제약조건에서 사용됨)
STATUS	VARCHAR2(8)	제약조건 활성화 여부 (ENABLED / DISABLED)
…		

Integrity Constraints 관련 정보

- CONSTRAINT_TYPE 컬럼
  - ✓ P: Primary key 제약조건
  - ✓ R: Foreign key 제약조건
  - ✓ U: Unique 제약조건
  - ✓ C: Check / NOT NULL 제약조건
- SEARCH_CONDITION 컬럼
  - ✓ Check 제약조건에서 사용된 조건식을 나타냄
    - 예: sal > 1000
  - ✓ NOT NULL 제약조건
    - 예: "ENAME" IS NOT NULL

Integrity Constraints 관련 정보

- USER_CONS_COLUMNS
  - 제약조건이 설정된 컬럼들에 관한 정보 저장
  - 일반적으로 USER_CONSTRAINTS view와 함께 사용됨 (1:N 관계)

컬럼명	데이터 타입	의미
OWNER	VARCHAR2(30)	제약조건의 소유자
CONSTRAINT_NAME	VARCHAR2(30)	제약조건 이름
TABLE_NAME	VARCHAR2(30)	제약조건이 적용되는 테이블 이름
COLUMN_NAME	VARCHAR2(4000)	제약조건이 설정된 컬럼 이름
POSITION	NUMBER	제약조건 정의에서 사용된 컬럼의 위치

Primary Key Constraints

- PRIMARY KEY
  - 테이블 내에서 행들을 서로 구별할 수 있도록 하는 컬럼(들의 집합)
  - PRIMARY KEY = NOT NULL + UNIQUE
    - NULL 값이나 중복되는 값을 입력 시 오류 발생
  - 예

```
CREATE TABLE dept02
  (deptno   NUMBER(2)  PRIMARY KEY,
   dname    VARCHAR2(15),
   loc      VARCHAR2(15));

CREATE TABLE dept03
  (deptno   NUMBER(2),
   dname    VARCHAR2(15),
   loc      VARCHAR2(15),
   CONSTRAINT dept03_deptno_pk  PRIMARY KEY (deptno));
```

CONSTRAINT *constraint_name* 생략됨

Primary Key Constraints

– 제약조건 확인

```
SELECT constraint_name, constraint_type, table_name
FROM USER_CONSTRAINT;
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME
FK_DEPTNO	R	EMP
PK_DEPT	P	DEPT
PK_EMP	P	EMP
DEPT03_DEPTNO_PK	P	DEPT03
SYS_C004611	P	DEPT02

이름이 자동 생성됨 ⟶

```
INSERT INTO dept03 VALUES (1, 'AA', 'AAAA');
INSERT INTO dept03 VALUES (1, 'BB', 'BBBB');        // error!
INSERT INTO dept03 VALUES (NULL, 'CC', 'CCCC');     // error!
```


Primary Key Constraints

– 복합 키 정의

- Table-level definition 만 가능
- 예

```
CREATE TABLE dept06
    (deptno   NUMBER(2),
     dname    VARCHAR2(15),
     loc      VARCHAR2(15),
     PRIMARY KEY (deptno, dname));
```


Primary Key Constraints

– 제약조건 확인

```
SELECT owner, constraints_name, constraints_type
FROM USER_CONSTRAINT
WHERE table_name='DEPT06';
```

OWNER	CONSTRAINT_NAME	CONSTRAINT_TYPE
SCOTT	SYS_C004607	P

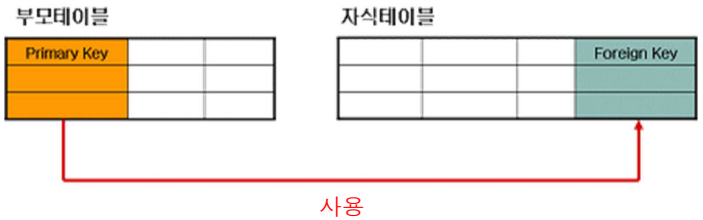
```
SELECT *
FROM USER_CONS_COLUMNS
WHERE table_name='DEPT06';
```

OWNER	CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	POSITION
SCOTT	SYS_C004607	DEPT06	DNAME	2
SCOTT	SYS_C004607	DEPT06	DEPTNO	1

Foreign Key Constraints

◆ 참조 무결성 제약조건(Referential Integrity Constraints)

– 주종 관계에 있는 테이블들에 대해, 부모 테이블의 기본 키(Primary Key)와 자식 테이블의 외래 키(Foreign Key)를 통해 참조 무결성 제약조건을 정의



Foreign Key Constraints

- 예: 모든 사원은 회사 내에 존재하는 부서에만 소속될 수 있음
  - 사원 테이블에 있는 부서번호(emp.deptno)의 값은 부서 테이블의 부서번호(dept.deptno) 값을 참조해야 함

➔ dept.deptno에는 Primary Key 또는 Unique 제약조건을 정의, emp.deptno에는 Foreign Key 제약조건을 정의

Foreign Key Constraints

dept

```
C:₩WINDOWS₩system32₩cmd.ex...
SQL> SELECT * FROM dept;

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

emp

```
C:₩WINDOWS₩system32₩cmd.exe - sqlplus SCOTT/TIGER
SQL> SELECT * FROM emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 80/12/17        800                    20
      7499 ALLEN      SALESMAN        7698 81/02/20       1600        300         30
      7521 WARD       SALESMAN        7698 81/02/22       1250        500         30
      7566 JONES      MANAGER         7839 81/04/02       2975                    20
      7654 MARTIN     SALESMAN        7698 81/09/28       1250       1400         30
      7698 BLAKE      MANAGER         7839 81/05/01       2850                    30
      7782 CLARK      MANAGER         7839 81/06/09       2450                    10
      7788 SCOTT      ANALYST         7566 87/04/19        100                    20
      7839 KING       PRESIDENT            81/11/17       5000                    10
      7844 TURNER     SALESMAN        7698 81/09/08       1500          0         30
      7876 ADAMS      CLERK           7788 87/05/23       1100                    20
      7900 JAMES      CLERK           7698 81/12/03        950                    30
      7902 FORD       ANALYST         7566 81/12/03       3000                    20
      7934 MILLER     CLERK           7782 82/01/23       1300                    10
14 개의 행이 선택되었습니다.
```

참조 무결성 보장

Foreign Key Constraints

◆ 제약조건 정의 예

- 먼저 deptno 컬럼을 기본 키로 갖는 부서 테이블을 생성

```
CREATE TABLE  dept07 (
    deptno   NUMBER(2) ,
    dname    VARCHAR2(15),
    loc      VARCHAR2(15),
    CONSTRAINT dept07_deptno_pk  PRIMARY KEY (deptno) );
```

- 사원 테이블 생성 시 deptno 컬럼을 외래 키로 지정(부서 테이블의 deptno 컬럼을 참조)

```
CREATE TABLE  emp07 (
    empno    NUMBER(4) PRIMARY KEY,
    ename    VARCHAR2(10),  sal  NUMBER(7,2),
    deptno   NUMBER(2)  REFERENCES dept07 (deptno) );
```

(deptno) 생략 가능

deptno 컬럼과 별도로 정의: FOREIGN KEY (deptno) REFERENCES dept07 (deptno)

Foreign Key Constraints

- 설정 확인

```
SELECT table_name, constraint_name, constraint_type, r_constraint_name
FROM USER_CONSTRAINTS
WHERE table_name LIKE '%07'
ORDER BY table_name;
```

TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE	R_CONSTRAINT_NAME
DEPT07	DEPT07_DEPTNO_PK	P	-
EMP07	SYS_C004613	P	-
EMP07	SYS_C004614	R	DEPT07_DEPTNO_PK

Foreign Key Constraints

- **ON DELETE CASCADE/SET NULL** option
  - 외래 키에 의해 참조되는 행이 삭제될 때 그것을 참조하는 행들도 같이 삭제하거나, 외래 키 컬럼 값을 NULL로 변경함

```
CREATE TABLE  emp07  (
    empno    NUMBER(4) PRIMARY KEY,
    ...
    deptno   NUMBER(2) REFERENCES dept07 (deptno)
                       ON DELETE CASCADE  -- 또는 SET NULL
);
```

→ dept07 테이블의 특정 행(부서) 삭제 시 emp07.deptno를 통해 그 행을 참조하는 행들(소속 사원들)이 있을 경우, 그 행들을 같이 삭제하거나(CASCADE), 그 행들의 detpno 값을 모두 NULL로 변경함(SET NULL)

Foreign Key Constraints

- 제약조건 검사 확인

```
INSERT INTO dept07 VALUES (1, 'AA', 'AAAA');
INSERT INTO dept07 VALUES (2, 'BB', 'BBBB');

INSERT INTO emp07 VALUES (1111, 'SMITH', 2000, 1);    // OK
INSERT INTO emp07 VALUES (2222, 'JAIN', 3000, 3);     // error!
```

```
ORA-02291: 무결성 제약조건(SCOTT.SYS_C004614)이 위배되었습니다- 부모 키가 없습니다
```

```
DELETE FROM dept07 WHERE deptno=1;                    // error!
```

```
ORA-02292: 무결성 제약조건(SCOTT.SYS_C004614)이 위배되었습니다- 자식 레코드가 발견되었습니다
```

- emp07.deptno에 대한 FK 정의 시 ON DELETE CASCADE option이 추가되었을 경우 → 위 오류가 발생하지 않고 emp07의 첫 번째 행이 같이 삭제됨

Unique Constraints

- UNIQUE
  - 컬럼의 모든 값이 유일해야 함
  - 주의: NULL은 중복해서 저장될 수 있음 (NULL은 값이 아님)
  - 예:

```
CREATE TABLE dept08
   (deptno  NUMBER  PRIMARY KEY,
    dname   VARCHAR2(15)  UNIQUE,
    loc     VARCHAR2(15),
    CONSTRAINT dept08_loc_uq UNIQUE (loc));

INSERT INTO dept08 VALUES (1, 'AA', 'AAAA');
INSERT INTO dept08 VALUES (2, 'BB', 'AAAA');      // error!
INSERT INTO dept08 VALUES (3, 'CC', NULL);        // OK
INSERT INTO dept08 VALUES (4, 'DD', NULL);        // OK
```


NOT NULL Constraints

- NOT NULL
  - 컬럼에 NULL이 저장될 수 없음
  - 예:

```
CREATE TABLE dept09
  (deptno   NUMBER(4)  PRIMARY KEY,
   dname    VARCHAR2(15)
            CONSTRAINT  dept09_dname_nn NOT NULL,
   loc      VARCHAR2(15));

INSERT INTO dept09 VALUES (1, NULL, 'AAAA');      // error!
```

생략 가능

Check Constraints

- CHECK
  - 컬럼 값에 대해 항상 만족해야 하는 일반적인 조건을 정의
  - 예

```
CREATE TABLE emp08
   (empno   NUMBER(4)  PRIMARY KEY,
    ename   VARCHAR2(15),
    sal     NUMBER(7, 2) ,
    CONSTRAINT emp08_sal_ck CHECK (sal BETWEEN 500 AND 5000));

INSERT INTO emp08 VALUES (1111, 'SMITH', 6000);      // error!
```


Check Constraints

- CHECK 제약 조건 설정 확인

```
SELECT table_name, constraint_name, constraint_type, search_condition
FROM USER_CONSTRAINTS
WHERE table_name = 'EMP08';
```

TABLE_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
EMP08	EMP08_SAL_CK	C	sal BETWEEN 500 AND 5000
EMP08	SYS_C004619	P	-

  - SEARCH_CONDITION 컬럼
    - ✓ CHECK 제약조건에서 사용된 조건식을 저장

제약조건의 추가 및 삭제

Table은 이미 추가되었음!

- 제약조건 추가
  - Primary key

```
ALTER TABLE dept
    ADD CONSTRAINT dept_deptno_pk PRIMARY KEY (deptno);
```

  - Foreign key

```
ALTER TABLE emp
    ADD CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
    REFERENCES dept (deptno) [ON DELETE CASCADE/SET NULL];
```

둘중 하나

  - Not Null

```
ALTER TABLE dept
    MODIFY loc  CONSTRAINT dept_dname_nn NOT NULL;
```

실습#2-2번문제

제약조건의 추가 및 삭제

- 제약조건 삭제
  - Primary key

```
ALTER TABLE dept DROP CONSTRAINT dept_deptno_pk ;
ALTER TABLE dept DROP CONSTRAINT dept_deptno_pk CASCADE ;
```

    - ✓ **CASCADE** option: 다른 테이블에 정의된 외래 키 제약조건 때문에 기본 키 제약조건의 삭제가 불가능할 경우, 그 외래 키 제약조건을 먼저 삭제하고 기본 키 제약조건을 삭제함
  - Foreign key

```
ALTER TABLE emp DROP FOREIGN KEY emp_deptno_fk;
```

제약조건 비활성화

- DISABLE / ENABLE
  - 제약조건을 삭제하지 않고 적용을 잠시 보류하는 기능

```
ALTER TABLE emp
    DISABLE CONSTRAINT emp_empno_pk;
```

✓ 제약조건을 비활성화 하여 테이블에 적용하지 않음

```
ALTER TABLE emp
    DISABLE CONSTRAINT emp_empno_pk CASCADE;
```

✓ **CASCADE** option: 다른 테이블의 외래 키에 의해 참조되어 비활성화가 불가능한 경우 그 외래 키 제약도 비활성화 시킴

```
ALTER TABLE emp
    ENABLE CONSTRAINT emp_empno_pk;
```

✓ 비활성화가 된 제약조건을 다시 활성화시킴

✓ 만일 중복된 값들이 존재할 경우 활성화가 불가능 함

Sequences

- Sequence란?
  - 일련의 증가 또는 감소하는 숫자 값들을 생성하는 자동 번호 생성기
  - 데이터베이스 내에 저장되어 여러 사용자가 공유 가능
  - 항상 유일한 값을 생성
    - 두 사용자가 같은 번호 값을 얻을 수 없음
  - 주로 테이블의 기본 키(primary key) 값을 자동 생성하기 위해 사용됨

Sequences

- 생성

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

생략 가능 (생략 시 초기값으로 설정)

  - INCREMENT BY n: sequence의 증가/감소 값 지정
    - 양수인 경우 증가 sequence, 음수인 경우 감소 sequence가 됨(default = 1)
  - START WITH n: sequence의 시작 값 지정
    - default 값은 MINVALUE(증가 sequence의 경우) / MAXVALUE(감소 sequence의 경우)
  - MAXVALUE n: sequence의 최대 값 지정
    - NOMAXVALUE의 경우 최대 값이 10^{27} (증가 sequence) 또는 -1(감소 sequence)로 설정됨
  - MINVALUE n: sequence의 최소 값
    - NOMINVALUE의 경우 최소 값이 1(증가 sequence) 또는 -10^{26}(감소 sequence) 로 설정됨
  - CYCLE | NOCYCLE: sequence 값의 순환 여부
  - CACHE n: n개의 sequence 값을 생성하여 메모리에 cache (default = 20)

Sequences

- 변경

```
ALTER SEQUENCE sequence_name
        [ INCREMENT BY n]
        [ MAXVALUE n | NOMAXVALUE ]
        ...
```

  - 주의: START WITH n은 재설정 불가 → 초기값은 변경 불가
- 삭제: **DROP SEQUENCE** *sequence_name*;
- 사용: currval, nextval pseudo-column 이용
  - *sequence_name*.**currval**
    - sequence의 현재 값을 반환
  - *sequence_name*.**nextval**
    - sequence의 다음 값을 생성 및 반환
    - 생성된 값은 currval 의 값이 됨
  - 주의: currval 값을 사용하기 전에 nextval 값을 먼저 생성해야 함

Sequences 관련 정보

- DBA_SEQUENCES (Data Dictionary View)
  - 데이터베이스 내의 모든 sequence 정보 포함

컬럼명	데이터 타입	의미
SEQUENCE_OWNER	VARCHAR2(30)	sequence 소유자
SEQUENCE_NAME	VARCHAR2(30)	sequence 이름
MIN_VALUE	NUMBER	sequence의 최소 값
MAX_VALUE	NUMBER	sequence의 최대 값
INCREMENT_BY	NUMBER	sequence의 증가/감소 값
CYCLE_FLAG	VARCHAR2(1)	sequence의 순환가능 여부
CACHE_SIZE	VARCHAR2(1)	sequence 값의 캐시 저장 개수
LAST_NUMBER	NUMBER	sequence의 마지막 값

- USER_SEQUENCES (Data Dictionary View)
  - 특정 사용자가 생성한 sequence 정보
  - DBA_SEQUENCES와 유사하나 SEQUENCE_OWNER 컬럼이 없음

Sequence 사용 예

- sequence 생성 예

```
SQL> conn scott/tiger
연결되었습니다.
SQL> create sequence deptno_seq
  2  increment by 10
  3  start with 10;
시퀀스가 생성되었습니다.

SQL> column sequence_name format a10;
SQL> select sequence_name, min_value, max_value, increment_by, cycle_flag
  2  from user_sequences;

SEQUENCE_N  MIN_VALUE  MAX_VALUE INCREMENT_BY CY
---------- ---------- ---------- ------------ --
DEPTNO_SEQ          1 1.0000E+27           10 N
```


Sequence 사용 예

- sequence 이용 예

```
SQL> select deptno_seq.currval from dual;
select deptno_seq.currval from dual
       *
1행에 오류:
ORA-08002: 시퀀스 DEPTNO_SEQ.CURRVAL은 이 세션에서는 정의 되어 있지
않습니다

SQL> select deptno_seq.nextval from dual;
  NEXTVAL
----------
        10
SQL> select deptno_seq.currval from dual;
  CURRVAL
----------
        10
```


Sequence 사용 예

- sequence 이용 예

```
SQL> select deptno_seq.nextval from dual;
  NEXTVAL
----------
        20
SQL> select deptno_seq.nextval from dual;
  NEXTVAL
----------
        30
SQL> select deptno_seq.nextval from dual;
  NEXTVAL
----------
        40
SQL> select deptno_seq.currval from dual;
  CURRVAL
----------
        40
```

Sequence 사용 예

◆ sequence 변경, primary key 컬럼 값 생성 예

```
SQL> create table dept15
  2  (deptno number(4) primary key,
  3  dname varchar(15));

테이블이 생성되었습니다.
SQL> alter sequence deptno_seq
  2  increment by 100;

시퀀스가 변경되었습니다.
SQL> insert into dept15 values (deptno_seq.nextval, 'AAAA');
1 개의 행이 만들어졌습니다.
SQL> insert into dept15 values (deptno_seq.nextval, 'BBBB');
1 개의 행이 만들어졌습니다.
SQL> insert into dept15 values (deptno_seq.nextval, 'CCCC');
1 개의 행이 만들어졌습니다.
SQL> insert into dept15 values (deptno_seq.nextval, 'DDDD');
1 개의 행이 만들어졌습니다.
```

140이 들어감

Sequence 사용 예

◆ sequence 변경, primary key 컬럼 값 생성 예

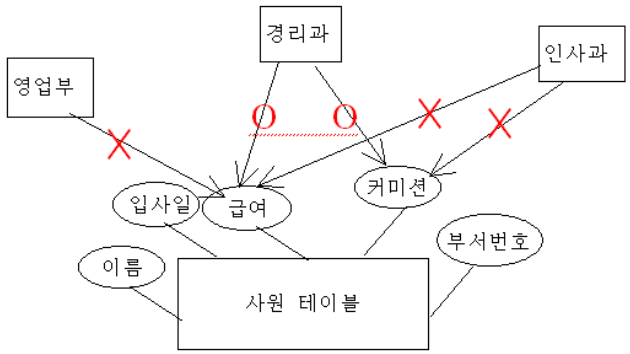
```
SQL> select * from dept15;

    DEPTNO DNAME
---------- ------------------------------
       140 AAAA
       240 BBBB
       340 CCCC
       440 DDDD

SQL>
```


Views

◆ 개념
- 물리적인 테이블들을 기반으로 정의되는 논리적 가상 테이블
  - 기반 테이블들로부터 파생된 객체
- 기반 테이블들에 대한 하나의 질의문(query)으로 표현
- View를 통해 기반 테이블들에 대한 사용자들의 접근을 제한함
- View에 대해 데이터를 삽입/수정/삭제하면 기반 테이블의 데이터가 변경됨

◆ 장점
- View를 이용하면 복잡한 질의를 단순하게 표현할 수 있음
- 전체 데이터의 일부만을 접근 가능하게 함으로써 보안성 향상

Views



- 같은 테이블에 대해 여러 view들을 정의하여 서로 다른 목적으로 사용하고 접근을 제어함

Views

emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	-	81/11/17	5000	-	10
7782	CLARK	MANAGER	7839	81/06/09	2450	-	10
7934	MILLER	CLERK	7782	82/01/23	1300	-	10
7902	FORD	ANALYST	7566	81/12/03	3000	-	20
7369	SMITH	CLERK	7902	80/12/17	800	-	20
7566	JONES	MANAGER	7839	81/04/02	2975	-	20
7900	JAMES	CLERK	7698	81/12/03	950	-	30
7844	TURNER	SALESMAN	7698	81/09/08	1500	0	30
7654	MARTIN	SALESMAN	7698	81/09/28	1250	1400	30
7521	WARD	SALESMAN	7698	81/02/22	1250	500	30
7499	ALLEN	SALESMAN	7698	81/02/20	1600	300	30
7698	BLAKE	MANAGER	7839	81/05/01	2850	-	30

emp_view30

EMPNO	ENAME	DEPTNO
7499	ALLEN	30
7521	WARD	30
7654	MARTIN	30
7698	BLAKE	30
7844	TURNER	30
7900	JAMES	30

```
CREATE VIEW emp_view30 AS
  SELECT empno, ename, deptno
  FROM emp
  WHERE deptno=30;
```

실행

```
SELECT ename
FROM emp_view30
WHERE ename like 'A%';
```

결과

ENAME
ALLEN

View 정의

- 형식

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view_name [(aliases)]
  AS subquery
  [WITH CHECK OPTION [CONSTRAINT constraint_name] ]
  [WITH READ ONLY]
```

- View의 종류
  - 단순 view – 하나의 기반 테이블로부터 정의됨
  - 복합 view – 여러 개의 기반 테이블들로부터 정의됨(join view)

View 정의

- 단순 view 예

```
CREATE VIEW emp_view10 (eno, name, dno) AS
  SELECT empno, ename, deptno
  FROM emp
  WHERE deptno=10;
```

```
CREATE VIEW dept_sum AS
  SELECT deptno, SUM(sal) sum_sal
  FROM emp
  GROUP BY deptno;
```

생략 시 오류 발생

View 정의

- 복합 view 예

```
CREATE VIEW emp_view_join AS
  SELECT e.empno, e.ename, d.dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```

desc emp_view_join

Table	Column	데이터 유형	길이
EMP_VIEW_JOIN	EMPNO	Number	-
	ENAME	Varchar2	10
	DNAME	Varchar2	14

SELECT * FROM emp_view_join;

EMPNO	ENAME	DNAME
7369	SMITH	RESEARCH
7499	ALLEN	SALES
7521	WARD	SALES
7566	JONES	RESEARCH
7654	MARTIN	SALES
7698	BLAKE	SALES

View 삭제, 변경

- View 삭제
  - 기반 테이블의 구조나 데이터에는 영향을 주지 않음

```
DROP VIEW view_name;
```

- View 정의 변경
  - view를 삭제한 후 재생성 해야 함
  - OR REPLACE option을 통해 view를 삭제하지 않고 view 정의 변경 가능

```
CREATE OR REPLACE VIEW emp_view_join AS
  SELECT e.ename, d.dname, d.loc
  FROM emp e, dept d
  WHERE e.deptno = d.deptno;
```


View Options

- FORCE
  - 일반적으로 기 존재하는 테이블에 대한 질의문으로 view를 생성해야 함
  - FORCE option 사용: 기반 테이블이 존재하지 않는 경우에도 view 생성 가능

```
CREATE FORCE VIEW employees_view AS
      SELECT * FROM employees;
```

- WITH CHECK OPTION
  - View 생성 시 WHERE 절에서 WITH CHECK OPTION 을 사용할 경우 그 조건에 사용된 컬럼 값을 view를 통해서 변경 불가능하게 함

```
CREATE OR REPLACE NOFORCE VIEW emp_chk20 AS
      SELECT empno, ename, deptno
      FROM emp
      WHERE deptno = 20 WITH CHECK OPTION;
// 이 view를 통해 emp 테이블의 deptno 컬럼 값 변경 불가
```


View Options

- WITH READ ONLY
  - View를 통해 기반 테이블의 데이터를 변경할 수 없도록 함

```
CREATE VIEW emp_chk30 AS
   SELECT empno, ename, deptno
   FROM emp
   WHERE e.deptno = 30 WITH READ ONLY;

UPDATE emp_chk30              // error!
SET ename = 'SYJ'
WHERE empno = 7788;
```


View 관련 정보

- USER_VIEWS (Data Dictionary View)

컬럼명	데이터 타입	의미
VIEW_NAME	VARCHAR2(30)	view 이름
TEXT_LENGTH	NUMBER	view 정의 텍스트의 길이
TEXT	LONG	view 정의 텍스트
…		

  - 사용 예

```
SELECT view_name, text, text_length
FROM   USER_VIEWS;
```

VIEW_NAME	TEXT	TEXT_LENGTH
EMP_VIEW30	SELECT empno, ename, deptno FROM emp WHERE deptno=30	52

참고: Inline View & Top-N Query

- Inline View
  - FROM 절 내부에 사용된 subquery는 일종의 view와 유사
- ROWNUM pseudo-column → 오라클의 자동으로 생성
  - 테이블(또는 view)로부터 행이 선택되는 순서대로 값이 결정됨 (1, 2, ...)

```
SELECT ROWNUM, ename, sal
FROM emp
ORDER BY sal;
```

FROM 다음에 ORDER BY

→ ROWNUM이 sal 값의 순서와 같지 않음

ROWNUM	ENAME	SAL
1	SMITH	800
10	JAMES	950
3	WARD	1250
5	MARTIN	1250
12	MILLER	1300
9	TURNER	1500

```
SELECT ROWNUM, ename, sal
FROM  (SELECT ename, sal
       FROM emp
       ORDER BY sal);
```

→ ROWNUM이 sal 값의 순서와 일치함

ROWNUM	ENAME	SAL
1	SMITH	800
2	JAMES	950
3	WARD	1250
4	MARTIN	1250
5	MILLER	1300
6	TURNER	1500

참고: Inline View & Top-N Query

- Top-N Query
  - Inline view 및 ROWNUM pseudo-column 이용
  - 예: 급여가 가장 높은 사원 5명은?

```
SELECT ROWNUM, ename, sal
FROM  (SELECT ename, sal
       FROM emp
       ORDER BY sal DESC)
WHERE ROWNUM <= 5;
```

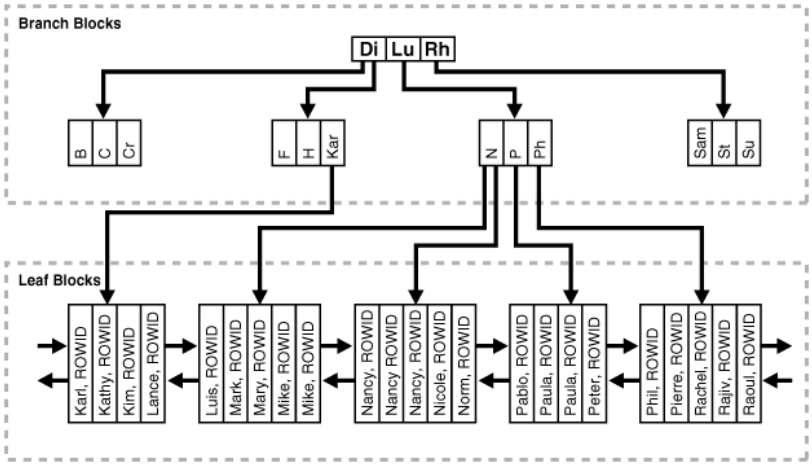
ROWNUM	ENAME	SAL
1	KING	5000
2	FORD	3000
3	JONES	2975
4	BLAKE	2850
5	CLARK	2450

Indexes

- Index란?
  - 테이블에 저장된 데이터의 빠른 검색을 위해 테이블의 특정 컬럼에 대해 생성하는 객체
    - 키 컬럼을 이용한 데이터 접근 경로 제공 (access method)
    - 키 컬럼 값을 이용하는 SQL 질의 실행 시 disk I/O의 수를 크게 줄임으로써 효율적인 질의 처리 가능
  - 일반적으로 B*-tree 구조를 가짐
    - Balanced tree – 일정한 검색 속도 제공
    - Tree의 각 node는 디스크의 block과 대응됨
    - Branch blocks: 트리의 내부 노드로서, (키 컬럼 값, 다른 노드에 대한 주소 값) 쌍들을 저장
    - Leaf blocks: 트리의 단말 노드로서, (키 컬럼 값, 그 값을 포함하는 레코드의 ROWID 값) 쌍들을 저장
      - ✓ ROWID를 통해 해당 레코드의 물리적 저장 위치를 찾을 수 있음
      - ✓ Leaf block들은 link를 통해 순서대로 연결됨
        - 모든 레코드들을 키 컬럼 값을 기준으로 순차적으로 접근 가능

Indexes

- B*-tree index의 내부 구조



Indexes

- Index 정의(생성)

  CREATE [UNIQUE] **INDEX** *index_name*
  ON *table_name* (*column_name_list*);

  - UNIQUE: unique index 생성

- Index 삭제

  DROP INDEX *index_name*;

Indexes

- Index 종류
  - 일반적인 index: single column, non-unique index
  - Unique Index
    - 키 컬럼에 저장된 값들의 유일성 보장
    - 예: create **unique** index emp3_idx1 on emp3(ename);
  - Composite Index
    - 동일 테이블의 여러 컬럼들에 대해 정의되는 index
    - 예: create index emp3_idx2 on emp3(**ename, deptno**);
  - Function-based Index
    - 테이블 내의 하나 이상의 컬럼들에 대한 산술식 또는 함수식으로 정의되는 index
    - 예: create index emp3_idx3 on emp3(**sal*12+comm**);

Indexes 관련 정보

- USER_INDEXES

컬럼명	데이터 타입	의미
INDEX_NAME	VARCHAR2(30)	index 이름
INDEX_TYPE	VARCHAR2(27)	index 타입
TABLE_OWNER	VARCHAR2(30)	index가 설정된 테이블 소유자
TABLE_NAME	VARCHAR2(30)	index가 설정된 테이블 이름
UNIQUENESS	VARCHAR2(9)	unique index 인지 여부
TABLESPACE_NAME	VARCHAR2(30)	index가 저장되는 tablespace 이름
…		

- USER_IND_COLUMNS

컬럼명	데이터 타입	의미
INDEX_NAME	VARCHAR2(30)	index 이름
TABLE_NAME	VARCHAR2(30)	index가 설정된 테이블 이름
COLUMN_NAME	VARCHAR2(9)	index가 설정된 컬럼 이름
COLUMN_POSITION	VARCHAR2(30)	index 정의에서 사용된 컬럼의 위치
…		

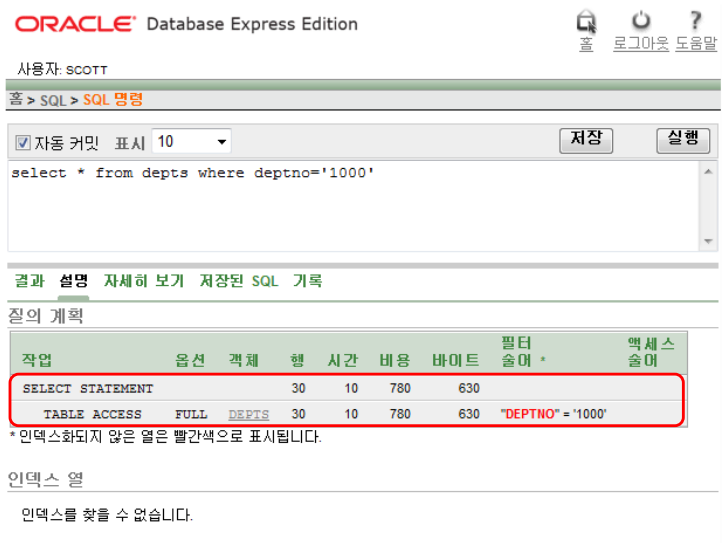
Index 사용 예

- Index 사용 및 효과
  - 테이블 생성 및 데이터 삽입

```
SQL> create table depts
  2  (deptno varchar2(10), dname varchar2(10), loc varchar2(10));

테이블이 생성되었습니다.
SQL> begin
  2     for i in 1..1000000 loop
  3        insert into depts values (i, 'AAA', 'BBB');
  4     end loop;
  5     commit;
  6  end;
  7  /

PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

Index 사용 예

- Index 생성 전의 질의 실행 계획(execution plan)



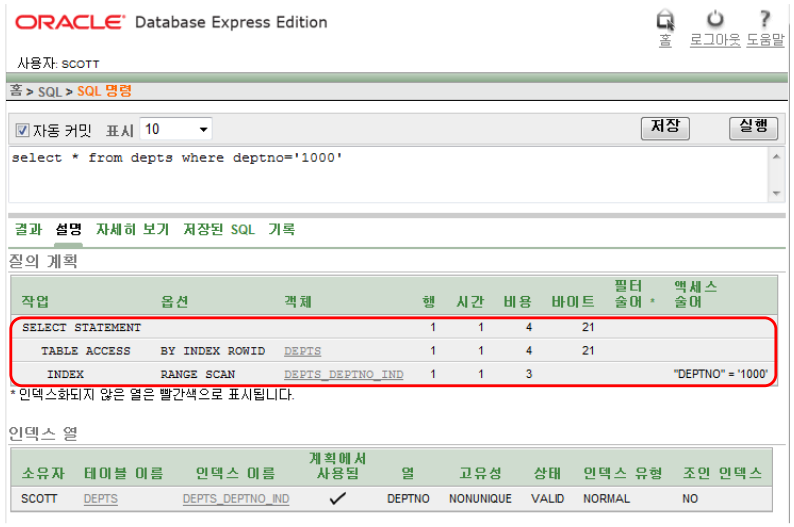
Index 사용 예

- Index 생성

```
SQL> create index depts_deptno_ind on depts(deptno);
인덱스가 생성되었습니다.

SQL> select index_name, table_name from user_indexes;
INDEX_NAME          TABLE_NAME
------------------- -------------------
PK_DEPT             DEPT
PK_EMP              EMP
DEPTS_DEPTNO_IND    DEPTS

SQL> select index_name, table_name, column_name from user_ind_columns;
INDEX_NAME          TABLE_NAME          COLUMN_NAME
------------------- ------------------- -------------------
PK_DEPT             DEPT                DEPTNO
PK_EMP              EMP                 EMPNO
DEPTS_DEPTNO_IND    DEPTS               DEPTNO
```


Index 사용 예

- Index 생성 후의 질의 실행 계획(execution plan)



Index 사용 예

- Index 사용이 필요한 경우
  - 해당 컬럼이 질의에서 WHERE 절의 선택 조건이나 join 조건으로 자주 사용될 때
  - 테이블에 저장된 행의 수가 많고 컬럼 값에 의한 검색 결과가 전체 행의 개수 대비 매우 적을 때
  - 컬럼 값이 UNIQUE 하거나 NULL 값이 많은 경우
- Index 사용이 부적절한 경우
  - 테이블의 데이터에 대한 입력/수정/삭제가 빈번히 발생하는 경우
    - Index도 함께 변경되어야 하므로 DML 문의 처리 성능이 저하됨
    - Index의 구조가 비효율적으로 바뀔 가능성이 크므로, 주기적인 재생성 필요