**[ 데이터 베이스 객체 ]**

* 테이블 : 데이터 저장 창고
* 인덱스 : 데이터 검색 효율을 높임
* 뷰 : 창고 안 들여다 보기 , 테이블과 매우 흡사한 오브젝트 임(가상 테이블)
* 호(시노님) : 객체에 별칭을 뷰여한다.
* 시퀀스 : 순번을 부여

1. **테이블**

**1.1 테이블?**

DB상에서 가장 기본이 되는 객체는 데이터를 저장하고 있는 테이블이다.

DB에서는 데이터들을 테이블을 통해서 관리하고 있다.

테이블은 엑셀시트처럼 생긴 2차원의 평면 구조를 가지고 있다.

테이블은 로우(행, row, 가로축)와 컬럼(열, column, 세로축)으로 구성된다.

창고에 비유한 용어 정리

|  |  |
| --- | --- |
| 창고 | DB 용어 |
| 총괄 책임자 | DBA |
| 구간 | 테이블 스페이스 |
| 담당자 | 사용자, 여기서는 스키마 소유자 |
| 컨테이너 박스 | 테이블 |
| 개별 물품 | 테이블에 저장되는 데이터 |
| 입고 및 출고 | INSERT, DELETE 등과 같은 DML |

논리적 데이터 모델링 :성격이 같은 데이터들을 분류하고 어떤 컨테이너를 몇 개나 만들지 결정

물리적 모델링 : 테이블 스페이스, 스키마 등 구체적인 설계를 하는 것

**1.2 테이블 만들기**

테이블은 가장 기본적이고 가장 대표적인 DB객체이며 DDL 문을 사용하여 생성함

|  |
| --- |
| **CREATE TABLES** employees  (  employee\_id NUMBER(6) NOT NULL,  first\_name VARCHAR2 (20 BYTE),  last\_name VARCHAR2 (25 BYTE) NOT NULL,  email VARCHAR2 (25 BYTE) NOT NULL,  phone\_number VARCHAR2(20 BYTE),  hire\_date DATE NOT NULL,  salary NUMBER(8,2),  manager\_id, NUMBER(6),  )  **TABLESPACE** example  **PCTUSED** 0  **PCTFREE** 10  **INITRANS** 1  **MAXTRANS** 255  **STORAGE** (  **INITIAL** 64K  **MINEXTENTS** 1  **MAXEXTENTS** 2147483645  **PCTINCREASE** 0  **BUFFER\_POOL** DEFAULT  )  **NOLOGGING**  **NOCOMPRESS**  **NOCACHE**  **NOPARALLEL**  **MONITORING;** |

DDL 문은 크게 두 부분으로 분리됨

- 첫번째 : 테이블 데이터 스키마

기본적인 생성 패턴 : **CREATE TABLE** 테이블명

컬럼1 컬럼1타입 …

테이블의 조회, 삽입, 삭제 등은 모두 로우 단위로 이루어지고, 테이블에 있는 컬럼들

이 모여 하나의 로우를 구성하게 된다.

테이블명 : 최대 30byte까지 가능(한글 15자)

컬럼명과 스키마, 사용자 이름도 30byte까지 사용 가능

한 테이블 최대 컬럼 개수 : 255개

컬럼 데이터 타입 : 오라클 기본 데이터 타입(문자, 숫자, 날짜)과 사용자 정의 데이터 타입으로 나눌 수 있다.

- 두번째 : DBA나 DB 설계자의 몫(생략 가능하며, 디폴트 값으로 설정됨)

**1.3 기본 데이터 타입(built-in datatype)**

크게 세가지로 구분(문자형, 숫자형, 날짜형)

**- 문자형 데이터**

|  |  |  |
| --- | --- | --- |
| 데이터 유형 | 설명 | 길이 |
| CHAR  [ size [BYTE] CHAR]) ] | 고정 길이의 문자형 데이터 타입  크기는 size 만큼의 byte 수 혹은 문자 개수가 됨 | 최대 크기 : 200byte  디폴트 : BYTE  Size 명시가 없을 경우 : 1 |
| VARCHAR2  [ size [BYTE | CHAR] ] | 가변 길이 문자형 데이터 타입  크기는 size 만큼의 byte 수 혹은 문자 개수가 됨 | 최대 크기 : 4000byte  디폴트 : BYTE  Size 생략 불가 |
| NOCHAR  [ (size) ] | 고정 길이의 유니코드 문자형 타입  크기는 UTF8 엔코딩의 경우 size 3\*byte, AL16UTF16 엔코딩의 경우 size\*2byte | 최대 크기 : 2000byte  Size 명시가 없을 경우 1 |
| NVARCHAR2  (size) | 가변 길이 유니코드 문자형 테이터 타입  크기는 최대 size값이 되며, UTF8의 경우 3배, AL16UTF16 엔코딩의 경우 2배의 byte가 됨 | 최대 크기 : 4000byte  Size 생략 불가 |
| LONG | 가변 길이의 문자형 데이터 타입 | 최대 크기 2GB |

\*) 일반적으로 가장 많이 사용되는 문자형 타입

CHAR : 데이터의 크기가 명확하게 정해져 있는 컬럼의 경우

VARCHAR2 : 데이터 크기가 변동적일 경우(대부분의 경우가 됨)

VARCHAR 타입은 VARCHAR2 타입과 동일하게 취급됨(오라클에서는 해당 타입 사용 자제)

ex) 고정 길이와 가변길이의 이해

|  |
| --- |
| **CREATE TABLE** char\_exam (  name1 **CHAR**(3 BYTE),  name2 **VARCHAR2**(3 BYTE) );  *CREATE TABLE succeeded*  **INSERT INTO** char\_exam **VALUES** ( ‘AA’, ‘AA’ );  *1 rows inserted* |
| **SELECT REPLACE**( name1, ‘ ‘ , ‘B’) nm1,  **REPLACE**(name2, ‘ ‘, ‘B’) nm2  **FROM** char\_exam; |
| [ 출력 결과 ]  NM1 NM2  AAB AA |

\*) REPLACE 함수 : 특정 문자를 다른 문자로 대체하는 역할

ex) 고정 길이와 가변길이의 이해

|  |
| --- |
| **CREATE TABLE** char\_exam1 (  Name3 **CHAR**(3 BYTE),  Name4 **CHAR**(3 CHAR ),  Name5 **CHAR**(3 ) );  *CREATE TABLE succeeded* |
| \*) name4 : 3글자의 크기를 갖음(byte 환산시 3byte나 6byte가 될 수 있다.) |
| **INSERT INTO** char\_exam1 (name3, name4) **VALUES** ( ‘AAA’, ‘AAA’ );  *1 rows inserted* |
| **INSERT INTO** char\_exam1 (name3) **VALUES** ( ‘홍길동’ );  *Error report;*  *SQL Error : ORA-12899: “HR”.”CHR\_EXAM1”.”NAMES3” 열에 대한 값이 너무 큼(실제:6, 최대값:3)* |
| **INSERT INTO** char\_exam1 (name4) **VALUES** ( ‘홍길동’ );  *1 rows inserted* |

**- 숫자형 데이터**

|  |  |  |
| --- | --- | --- |
| 데이터 유형 | 설명 | 길이 |
| BINARY\_FLOAT | 32bit 부동 소수 | 4byte |
| BINARY\_DOUBLE | 64bit 부동 소수 | 8byte |
| NUMBER  [ (prec | prec, scale) ] | 가변 숫자 타입  설정한 값(1~38)은 전체 자리수이며,  Scale로 설정한 값(-84~127)은 소수점 이하 자리수를 의미 | 최대값은 21byte |

\*) 숫자는 정수, 실수로 나눌 수 있다.

\*) 가장 많이 사용되는 NUMBER 타입은 정수를 포함한 모든 실수를 표현하고 저장한다.

ex) NUMBER 사용예

|  |  |
| --- | --- |
| **CREATE TABLE** num\_exam1 (  n1 **NUMBER**,  n2 **NUMBER** (9),  n3 **NUMBER** (9,2 ),  n4 **NUMBER** (9,1 ),  n5 **NUMBER** (7 ),  n6 **NUMBER** (7,-2 ),  n7 **NUMBER** (6 ),  n8 **NUMBER** (3,5 ),  *CREATE TABLE succeeded* | 자리수 제한이 없슴, 입력값의 손실이 없이 저장됨  전체 자리수 9, 소수점0, 소수점 첫 번째 자리에서 반올림  전체 자리수 9, 소수점 이하 2, 손실없이 저장  소수점 두 번째 자리에서 반올림  소수점 첫 번째 자리에서 반올림, 결과적으로 n2와 동일  -2, 소수점 기준으로 오른쪽이 아닌 왼쪽 두 번째 자리에서 반올림 |
| **INSERT INTO** num\_exam1 (n1, n2, n3, n4, n5, n6)  **VALUES** ( 1234567.89, 1234567.89, 1234567.89, 1234567.89, 1234567.89, 1234567.89 );  *1 rows inserted* | |
| **SELECT** n1, n2, n3, n4, n5, n6  **FROM** num\_exam1 | |
| [ 출력 결과 ]  N1 n2 n3 n4 n5 n6  ----------- ----------- ----------- ----------- ----------- -----------  1234567.89 1234568 1234567.89 1234567.9 1234568 1234600 | |

\*) BINARY\_FLOAT 와 BINARY\_DOUBLE 타입은 부동 소수점 수를 표현한다.

NUMBER 타입은 10진법으로 정밀도를 제어하지만 해당 타입은 2진법으로 정밀도를 제어하여 사용한다. 따라서 NUMBER 보다 정밀도가 낮게 된다.(why? 0~1로 표현해야 하기 때문)

하지만 저장공간을 덜 차지하는 장점을 가지고 있슴

\*) 오라클에서 사용가능한 숫자 타입

오라클 지원 숫자 타입 : NUMBER, BINARY\_FLOAT, BINARY\_DOUBLE

ANSI 표준 숫자 타입 : INTEGER, FLOAT 🡺 오라클에서는 NUMBER로 타입 변환하여 저장

함

**- 날짜형 데이터**

|  |  |  |
| --- | --- | --- |
| 데이터 유형 | 설명 | 길이 |
| DATE | 고정 길이의 날짜 시간 데이터  BC 4712년 1월1일로부터 9999년 12월 31일까지 표현 가능 | 7byte이며 표현형태는 NLS\_DATE\_FORMAT 파라미터에 명시된 값을 따름  (ex) DD-MON-RR, 01-JAN-99) |
| INTERVAL YEAR  [ (yr\_prec) ] TO MONTH | 연도와 월 형태로 기간을 표현  yr\_prec : 날짜와 연도의 최대 자리수  범위 : 0~9  디폴트 : 2 | 5byte |
| INTERVAL DAY  [(day\_prec)] TO SECOND  [(frac\_sec\_prec)] | 요일, 시, 분, 초 형태로 기간을 표현  Day\_prec는 날짜 자리수  frac\_sec\_prec는 날짜의 초 자리수  범위 : 0~9  디폴트 : day\_prec : 2  frac\_sec\_prec : 6 | 11byte |
| TIMESTAMP  [((frac\_sec\_prec)) | 밀리초까지 표현 가능  frac\_sec\_prec : 초 정보의 정밀도  범위 : 0~9  디폴트 : 6 | precision에 따라 7~11byte로 디폴트 값은 NLS\_TIMESTAMP\_FORMAT에 명시된 값을 따름 |
| TIMESTAMP  [(frac\_sec\_prec)] WHTH  TIME ZONE | 시간대와 더불어 날짜와 시간 표현  시간대 : ‘-5:0’ 나 ‘US/Pacific’  Frac\_sec\_prec 는 TIMESTAMP | 13byte로 디폴트값은 NLS\_TIMESTAMP\_TZ\_FORMAT 파라미터에 명시된 값을 따름 |
| TIMESTAMP  [(frac\_sec\_prec)] WITH  LOCAL TIME ZNOE | TIMESTAMP WITH TIME ZONE 과 비슷하나 저장시점에는 DB 시간대를 준수하거나 조회시에는 조회하는 클라이언트의 시간대로 표현됨  Frac\_sec\_prec는 TIMESTAMP | Frac\_sec\_prec에 따라 7~11byte이며,  디폴트값은 NS\_TIMESTAMP\_FORMAT 파라미터에 명시된 값을 따름 |

\*) DATE 타입이 가장 많이 사용됨

ex) 사용예

|  |
| --- |
| **CREATE TABLE** date\_exam1 (  date1 **DATE**);  *CREATE TABLE succeeded*  **INSERT INTO** date\_exam1 (date1) **VALUES** (SYSDATE );  *1 rows inserted*  **SELECT \*** **FROM** date\_temp1; |
| [출력 결과]  DATE1  -------------------------------  10/08/05 |
| SYSDATE : 오라클에서 제공하는 시스템의 현재 날짜 정보를 가지고 있슴  출력 형식은 NLS\_DATE\_FORMAT 에 설정된 값으로 출력됨 |
| **ALTER SESSION SET** NLS\_DATE\_FORMAT = ‘YYYY-MM-DD’);  *ALTER SESSION SET succeededt;*  **SELECT \*** **FROM** date\_temp1; |
| 출력 결과]  DATE1  -------------------------------  2010- 08-05 |

ex) 사용예

|  |
| --- |
| **CREATE TABLE** date\_exam2 (  time1 **TIMESTAMP,**  time2 **TIMESTAMP WITH TIME ZONE**,  time3 **TIMESTAMP WITH LOCAL TIME ZONE** );  *CREATE TABLE succeeded*  **INSERT INTO** date\_exam2 **VALUES** (SYSDATE , SYSDATE, SYSDATE);  *1 rows inserted* |
| **ALTER SESSION SET** NLS\_DATE\_FORMAT = ‘YYYY-MM-DD HH:MI:SS.FF’);  *ALTER SESSION SET succeededt;*  **SELECT \*** **FROM** date\_temp2; |
| 출력 결과]  TIME1 TIME2  ------------------------------- ------------------------------------------------------  2010- 08-05 09:41:58.000000000 10/ 08/05 09:41:58.000000000  TIME3  ------------------------------------  ASIA/SEOUL 2010-08-05 09:41:58.000000000 |

\*) TIMESTAMP 는 DATE 타입보다 확장된 날짜형 데이터 타입으로 년, 월, 일, 시, 분, 초

정보 값을 저장한다.

\*) TIMESTAMP WHIT TIME ZONE 타입은 각 지역시간과 그리니치 표준시와의 차이를 포함

한 TIMESTAMP의 변형된 데이터 타입이다.

🡺 오라클에서 사용 가능한 ANSI 데이터 타입

|  |  |
| --- | --- |
| ANSI | Oracle 변환 타입 |
| CHARACTER(n)  CHAR(n) | CHAR(n) |
| CHARACTER VARYING(n)  CHAR VARYING(n) | CARCHAR(n) |
| NATIONAL CHARACTER(n)  NATIONAL CHAR(n)  NCHAR(n) | NCHAR(n) |
| NATIONAL CHARACTER VARYING(n)  NATIONAL CHAR VARYING(n)  NCHAR VARYING(n) | NVARCHAR2(n) |
| NUMBERIC(p,s)  DECIMAL(p,s)(a) | NUMBER(p,s) |
| INTEGET  INT SMALLINT | NUMBER(p,s) |
| FLOAT(b)  DOUBLE PROCISION©  REAL(d) | NUMBER |

ex) 사용예

|  |
| --- |
| **CREATE TABLE** ansi\_exam (  tmp\_coll **IINTEGER**);  *CREATE TABLE succeeded*  **DECS** ansi\_oracle; |
| [출력결과]  이름 널? 유형  TMP\_COL1 NUMBER(38) |

**1.4 컬럼 속성(무결성 제약조건)**

무결성 제약조건의 예

- 권한이 있는 사람만 해당 데이터를 볼 수 있다.

- 데이터의 정확성을 보장한다.

🡺 제대로 된 데이터들이 올바르게 저장될 수 있도록 하기 위해 DB측에서 제공되는 기능을 통

틀어 말함

( NULL, UNIQUE 키, 기본키(Primary Key), 외래키(Foreign Key), CHECK, 디폴트)

**NULL**

데이터가 없음을 의미

컬럼이 NULL값을 허용하는지 허용하지 않는지를 정해줄 수 있다.

테이블 생성 스크립트에서 데이터 타입 다음에 명시(NULL or NOT NULL)

명시하지 않을 경우 디폴트값은 NULL임

ex)

|  |
| --- |
| **CREATE TABLE** null\_test(  null\_col1 **VARCHAR2**(20) **NOT NULL**,  null\_col2 **VARCHAR2**(20) **NULL**,  null\_col3 **VARCHAR2**(20) );  *CREATE TABLE succeeded*  **INSERT INTO** null\_test ( null\_col1, null\_col2 ) **VALUES** (‘AA’, ‘BB’);  *1 row inserted*  **INSERT INTO** null\_test( null\_col2, null\_col3) **VALUES** (‘cc’, ‘dd’);  *SQL Error : ORA-01400: NULL을 (“DEVELOPER1”.”NULL\_TESTS”.”NULL\_COL1”) 안에 삽입할 수 없습니다.* |
| **INSERT INTO** null\_test( null\_col1, null\_col2, null\_col3) **VALUES** (‘TT’, ‘SS’, NULL);  *1 row inserted*  **INSERT INTO** null\_test( null\_col1, null\_col2, null\_col3) **VALUES**(‘TT’, ‘SS’, ‘ ‘);  *1 row inserted* |

\*) NULL 값을 직접 넣기 위해서는 ‘NULL’ 이나 ‘ ‘을 넣으면 된다.

\*) NULL은 공백(‘ ‘) 과 다르다. 공백은 공백(SPACE)이라는 값이 있다.

**UNIQUE 키**

데이터를 유일하게 식별하기 위한 무결성 제약조건 중 하나

중복값을 허용하지 않는다.

단일 컬럼을 UNIQUE키로 할 수도 있고, 한 개 이상의 컬럼도 UNIQUE 키로 만들 수 있다.

ex)

|  |
| --- |
| **CREATE TABLE** unique\_test(  uni\_col1 **VARCHAR2**(10) **UNIQUE** **NOT NULL**,  uni\_col2 **VARCHAR2**(10) **UNIQUE**,  uni\_col3 **VARCHAR2**(10) ) **NOT NULL**,  uni\_col4 **VARCHAR2**(10) **NOT NULL**,  **CONSTRAINTS** uni\_tmp\_uk UNIQUE ( uni\_col3, uni\_col4) );  *CREATE TABLE succeeded*  **INSERT INTO** unique\_test ( uni\_col1,uni\_col2, uni\_col3, uni\_col4 ) **VALUES** (‘A1’, ‘B1’, ‘C1’, ‘D1’ );  *1 row inserted*  **INSERT INTO** unique\_test ( uni\_col1,uni\_col2, uni\_col3, uni\_col4 ) **VALUES** (‘A2’, ‘B2’, ‘C2’, ‘D2’ );  *1 row inserted.* |
| **UPDATE** unique\_test  **SET** uni\_col1 = ‘A1’;  **WHERE** uni\_col2 = ‘B2’;  *SQL Error : ORA-00001 : 무결성 제약조건(DEVELOPER1.SYS\_C006725)에 위배됩니다.* |
| **INSERT INTO** unique\_test ( uni\_col1,uni\_col2, uni\_col3, uni\_col4 ) **VALUES** (‘A3’, ‘’, ‘C3’, ‘D3’ );  *1 row inserted*  **INSERT INTO** unique\_test ( uni\_col1,uni\_col2, uni\_col3, uni\_col4 ) **VALUES** (‘A4’, ‘’, ‘C4’, ‘D4’ );  *1 row inserted.* |

\*) UNIQUE 키는 NULL을 허용한다.

\*) NULL은 값을 없음을 의미하므로 UNIQUE 키 비교 대상에서 제외된다. 따라서 2개 이상 저장

가능하다.

\*) NULL 대신 공백을 저장하게 되면 중복 사용시 무결성 에러가 발생한다.

\*) 일반적으로 UNIQUE 속성을 가진 컬럼은 NOT NULL 속성을 주는 것이 좋다.

\*) uni\_col3 와 uni\_col4 컬럼은 두 컬럼의 조합이 UNIQUE KEY가 걸려있다.

개별적으로 동일한 값이 들어갈 수 있으나 조합 값은 달라야 한다.(ex) 지역이 다른 전화번호)

**기본키(Primary Key)**

UNIQUE와 동일하게 한 테이블에 있는 데이터들을 유일하게 식별하기 위한 무결성 제약조건이

다.

다른점은 기본 키가 걸린 컬럼들은 반드시(자동으로) NOT NULL 속성을 가진다.

(기본키 = UNIQUE + NOT NULL)

\*) 기본키와 UNIQUE 키 비교

|  |  |  |
| --- | --- | --- |
| 비교 내용 | 기본 키 | UNIQUE 키 |
| NULL허용 | 허용하지 않음 | 허용 |
| 컬럼 수 | 1개 이상 | 1개 이상 |
| 유일한 값 식별 | 가능 | 가능 |
| 테이블당 개수 | 오직1개만 생성 가능 | 여러 개 생성 가능 |
| 복합 키 생성 | 32개 컬럼까지 가능 | 32개 컬럼까지 가능 |
| 컬럼 데이터 타입 | LOB, LONG, LONG RAW, VARRAY,  NESTED TABLE, BFILE, REF,  TIMESTAMP WITH ZONE, 사용자 정의 타입 컬럼은 생성할 수 없다. | LOB, LONG, LONG RAW, VARRAY,  NESTED TABLE, OBJECT, REF,  TIMESTAMP WITH ZONE, 사용자 정의 타입 컬럼은 생성할 수 없다. |

\*) 기본키 정의는 테이블 생성 시 두 가지 방법으로 표현한다.

인라인 표현 : employee\_id **NUMBER**(6,0) **PRIMARY KEY**

아웃라인 표현 : employee\_id **NUMBER**(6,0),

**CONSTRAINT** “employee\_pk” **PRIMARY KEY**(employee\_id)

\*) 테이블 생성 후 기본 키 정의

**ALTER TABLE** 테이블명

**ADD CONSTRAINTS** “기본 키 이름” **PRIMARY KEY** (컬럼);

**외래키(Foreign Key)**

관계형 데이터베이스에서 관계(Relationship)이라는 용어는 테이블 간의 관계를 의미한다.

테이블 간의 관계는 공통값을 가진 컬럼을 통해 이루어진다.

기본키 : DEPARTMENTS 테이블의 Department\_id

외래키 : EMPLOYEES 테이블의 Department\_id

\*) DEPARTMENTS 테이블 : 부모 테이블 / EMPLOYEES 테이블 : 자식 테이블

\*) EMPLOYEES 테이블의 EMPLOYEES 테이블의 Department\_id 저장시 Department\_id 에 존재하

지 않는 값을 입력할 경우 저장이 되지 않는다. 즉 외래 키에 의해 DBMS 단에서 무결성을 확보

하게 된다.

\*)생성 구문 형식1(테이블 생성전 : 인라인/아웃라인)

컬럼명 **CONSTRAINTS** “외래키 이름”

**REFERENCES** 참조 테이블명 (참조 컬럼) ;

\*) 생성 구문 형식2(테이블 생성후)

**ALTER TABLE** 테이블명

**ADD CONSTRAINTS** “외래키 이름” **FOREIGN KEY** (컬럼)

**REFERENCES** 참조 테이블명( 참조 컬럼) ;

ex) 생성 예

|  |
| --- |
| department\_id **CONSTRAINT** fk\_deptno  **REFERENCES** departments (department\_id) ); /\* 인라인 \*/ |
| department\_id,  **CONSTRAINT** fk\_deptno  **FOREIGN KEY** (department\_id)  **REFERENCES** departments (department\_id) /\* 아웃라인 \*/ |
| **ALTER TABLE** employees  **ADD** **CONSTRAINT** fk\_deptno **FOREIGN KEY** ( department\_id )  **REFERENCES** departments (department\_id); /\* 테이블 생성 후 \*/ |

\*) 외래 키 제약사항

- 외래 키 생성 전 반드시 부모 테이블이 먼저 생성되어야 함

- 참조하는 부모 테이블의 컬럼은 반드시 기본 키 혹은 UNIQUE 키여야 함

- 한 개 이상의 컬럼으로 외래 키를 생성할 수 있다. 복합 외래키의 경우 참조하는 부모

테이블의 기본 키나 UNIQUE 키도 반드시 복합 키가 되어야 한다.

- 자식 테이블에 존재하는 값을 부모테이블에서 삭제할 수 없다.

: INSERT시 : 부모 테이블 데이터 입력 🡺 자식 테이블 데이터 입력

: DELETE시 : 자식 테이블 데이터 삭제(수정) 🡺 부모 테이블 데이터 삭제(수정)

**CHECK**

입력되는 값을 체크하여 일정한 조건에 해당되는 값만 입력될 수 있게 하는 제약조건

Ex) 성별을 나타내는 gender라는 컬럼에 ‘남성’ ‘여성’ 만 입력되게 할 수 있다.

|  |
| --- |
| gender **VARCHAR2**(10) **NOT NULL ,**  **CONSTRAINT** check\_gender **CHECK** ( gender **IN** (‘남성’, ‘여성’) ) |
| gender **VARCHAR2**(10) **NOT NULL**  **CONSTRAINT** check\_gender **CHECK** ( gender **IN** (‘남성’, ‘여성’) ) |

\*) 위의 두 가지 방법으로 추가 가능

Ex) CHECK 제약조건 위배 예제

|  |
| --- |
| CREATE TABLE check\_test (  gender **VARCHAR2**(10) **NOT NULL**  **CONSTRAINT** check\_gender **CHECK** ( gender **IN** (‘남성’, ‘여성’) ) );  *CREATE TABLE succeeded* |
| INSERT INTO check\_test VALUES(‘무성’)  *SQL Error : ORA-02290: 체크 제약조건(HR, CHECK\_GENDER)이 위배되었습니다.* |

**디폴트**

데이터 제약조건에 포함되지는 않지만 컬럼 단위로 설정되는 속성이다.

컬럼에 특정 값을 디폴트값으로 설정하면 테이블에 데이터를 입력할 때 해당 컬럼에 값을 입력

하지 않을 경우 디폴트 설정값이 자동 입력된다.

\*) 설정 방법

컬럼 타입 다음에 ‘DEFAULT 디폴트값 ‘ 명시

NULL, NOT NULL 앞에 위치

|  |
| --- |
| Gender **VARCHAR2**(10) **DEFAULT** ‘남성’ NOT NULL  **CONSTRAINT** check\_gender **CHECK** ( gender IN (‘남성’, ‘여성’ ) ),  Insert\_date DATE **DEFAULT** SYSDATE,  … |

\*) 설정 방법(테이블 생성 후)

|  |
| --- |
| **ALTER TABLE** check\_test  **MODIFY** gender **DEFAULT** ‘남성’;  *ALTER TABLE check\_test succeeded*  **ALTER TABLE** check\_test  **ADD** insert\_date **DATE DEFAULT SYSDATE**;  *ALTER TABLE check\_test succeeded*  **ALTER TABLE** check\_test  **ADD** test **VARCHAR2(10) NULL**;  *ALTER TABLE check\_test succeeded*  **INSERT INTO check\_test (test) VALUES (‘AA’ );**  *1 rows inserted*  **SELECT \*FROM** check\_test; |
| [출력 결과]  GENDRE INSERT\_DATE TEST  ---------------- ------------------------- ----------------------  남성 10/08/05 AA |

\*) 추가적인 문법

테이블 생성시 컬럼명에 “ “로 표시가능 단, ‘ ‘ 는 안됨

EMPLOYEES 테이블의 정식 명칭은 HR.EMPLOYEES 이다.

HR은 HR 스키마에 있는 EMPLOYEES 테이블이라는 뜻임, 하지만 HR 사용자로 로그인하게

되면 HR을 생략해도 된다. 만약, 다른 사용자로 로그인한 상태에서는 HR.EMPLOYEES 이름으로

접근해야 한다.

|  |
| --- |
| **SELECT \* FROM** employees;  *SQL Error : ORA-00942 : 테이블 또는 뷰가 존재하지 않습니다.* |
| **SELECT \* FROM** hr.employees;  …  *107 rows selected* |

**1.5 테이블 복사**

기존에 있는 테이블을 그대로 복사할 수 있다.

반드시 테이블 이름이 중복되지 않게 만들어야 한다. 단 스키마가 다르면 상관없다.

|  |
| --- |
| **[구문 형식]**  **CREATE TABLE “**스키마명.테이블명**” AS**;  **SELECT** 컬럼 리스트  **FROM** 원본 테이블명; |
| **CREATE TABLE** emp\_copy1 **AS**;  **SELECT** \*  **FROM** EMPLOYEES;  *CREATE TABLE succeeded* |

\*) 원본 테이블의 데이터까지 그대로 복사하게 된다.

\*) 제약 사항

원본 테이블의 인덱스는 복사하지 않는다.

원본 테이블의 여러 가지 제약사항 중 NULL, NOT NULL 속성만 복사한다.

LONG 타입 컬럼은 복사할 수 없다.

**1.6 테이블 제거**

|  |
| --- |
| **[구문 형식]**  **DROP TABLE “**스키마명.테이블명**” [CASCADE CONSTRAINTS ];** |

\*) DROP TABLE 은 DDL 문이므로 ROLLBACK에 의해 복원될 수 없다.

\*) 테이블을 생성한 소유자만 해당 테이블을 제거할 수 있다.

SYS, SYSTEM, 그리고 DROP ANY TABLE 권한을 가진 사용자들은 모든 테이블을 제거할

수 있다

\*) 참조 제약조건(외래 키)들까지 자동으로 삭제시키려면 CASCADE CONSTRAINTS 옵션을 붙

여야 한다.

만약 참조 제약조건들이 존재하는 상황에서 옵션 없이 삭제한다면 오류가 발생한다.

* 테이블 관련 실습

|  |
| --- |
| ■ 다음과 같은 내용으로 테이블을 생성해 보자. (HR 사용자로 로그인 하도록 한다)  1. 테이블 명 : STAR\_WARS (영화 정보를 저장한다)  컬럼 : - EPISODE\_ID : 에피소드 아이디로써, 숫자형 타입으로 기본 키가 된다.  - EPISODE\_NAME : 에피소드에 따른 영화 제목이며, 가변 길이 문자형 타입(50 BYTE)이다.  - OPEN\_YEAR : 개봉년도로써, 역시 숫자형 타입이다.    2. 테이블 명 : CHARACTERS ( 등장인물 정보를 저장한다)  컬럼 : - CHARACTER\_ID : 등장인물 아이디로써, 숫자형 타입(5자리)으로 기본 키가 된다.  - CHARACTER\_NAME : 등장인물 명으로 가변 길이 문자형 타입(30 BYTE)이다.  - MASTER\_ID : 등장인물이 제다이일 경우 마스터 아이디 값을 가진다. 역시 숫자형(5자리)이다.  - ROLE\_ID : 등장인물의 역할 아이디로써, INTEGER 타입이다.  - EMAIL : 등장인물의 이메일 주소로 가변길이 문자형 타입(40 BYTE)이다.  3. 테이블 명 : - CASTING ( 등장인물과 실제 배우의 정보를 저장한다)  컬럼 : - EPISODE\_ID: 에피소드 아이디로써, 숫자형 타입(5자리)으로 기본 키가 된다.  - CHARACTER\_ID: 등장인물 아이디로써, 숫자형 타입(5자리)이며 참조 키가 된다.  - REAL\_NAME : 등장인물의 실제 이름으로, 가변길이 문자형 타입(30 BYTE)이다.    <<정답>> --------------------------------------------------------------------------------------------------  CREATE TABLE STAR\_WARS (  EPISODE\_ID NUMBER(5,0) NOT NULL,  EPISODE\_NAME VARCHAR2(50 BYTE),  OPEN\_YEAR NUMBER(5,0),  CONSTRAINT STAR\_WARS\_PK PRIMARY KEY (EPISODE\_ID));  -----------------------------------------------------------------------------------------------------------  <<정답>> --------------------------------------------------------------------------------------------------  CREATE TABLE CHARACTERS (  CHARACTER\_ID NUMBER NOT NULL,  CHARACTER\_NAME VARCHAR2(30 BYTE),  MASTER\_ID NUMBER(5,0),  ROLE\_ID NUMBER(5,0),  EMAIL VARCHAR2(40 BYTE),  CONSTRAINT CHARACTERS\_PK PRIMARY KEY (CHARACTER\_ID));  -----------------------------------------------------------------------------------------------------------  <<정답>> --------------------------------------------------------------------------------------------------  CREATE TABLE CASTING (  EPISODE\_ID NUMBER(5,0) NOT NULL,  CHARACTER\_ID NUMBER(5,0) NOT NULL,  REAL\_NAME VARCHAR2(30 BYTE),  CONSTRAINT CASTING\_PK PRIMARY KEY (EPISODE\_ID, CHARACTER\_ID));  4. INSERT 문을 사용하여 STAR\_WARS 테이블에 다음과 같이 데이터를 저장해보자.  EPISODE\_ID EPISODE\_NAME OPEN\_YEAR  ----------- --------------------------------------- --------------  4 새로운 희망(A New Hope) 1977  5 제국의 역습(The Empire Strikes Back) 1980  6 제다이의 귀환(Return of the Jedi) 1983  1 보이지 않는 위험(The Phantom Menace) 1999  2 클론의 습격(Attack of the Clones) 2002  3 시스의 복수(Revenge of the Sith) 2005  <<정답>> --------------------------------------------------------------------------------------------------  INSERT INTO STAR\_WARS ( EPISODE\_ID, EPISODE\_NAME, OPEN\_YEAR)  VALUES ( 4, '새로운 희망(A New Hope)', 1977);  INSERT INTO STAR\_WARS ( EPISODE\_ID, EPISODE\_NAME, OPEN\_YEAR)  VALUES ( 5, '제국의 역습(The Empire Strikes Back)', 1980);  INSERT INTO STAR\_WARS ( EPISODE\_ID, EPISODE\_NAME, OPEN\_YEAR)  VALUES ( 6, '제다이의 귀환(Return of the Jedi)', 1983);  INSERT INTO STAR\_WARS ( EPISODE\_ID, EPISODE\_NAME, OPEN\_YEAR)  VALUES ( 1, '보이지 않는 위험(The Phantom Menace)', 1999);  INSERT INTO STAR\_WARS ( EPISODE\_ID, EPISODE\_NAME, OPEN\_YEAR)  VALUES ( 2, '클론의 습격(Attack of the Clones)', 2002);  INSERT INTO STAR\_WARS ( EPISODE\_ID, EPISODE\_NAME, OPEN\_YEAR)  VALUES ( 3, '시스의 복수(Revenge of the Sith)', 2005);  COMMIT;  -----------------------------------------------------------------------------------------------------------  5. CHARACTERS 테이블에 다음의 데이터를 저장해보자.  CHARACTER\_ID CHARACTER\_NAME EMAIL  --------------- -------------------- ------------------------  1 루크 스카이워커 luke@jedai.com  2 한 솔로 solo@alliance.com  3 레이아 공주 leia@alliance.com  4 오비완 케노비 Obi-Wan@jedai.com  5 다쓰 베이더 vader@sith.com  6 다쓰 베이더(목소리) Chewbacca@alliance.com  7 C-3PO c3po@alliance.com  8 R2-D2 r2d2@alliance.com  9 츄바카 Chewbacca@alliance.com  10 랜도 칼리시안  11 요다(목소리) yoda@jedai.com  12 다스 시디어스  13 아나킨 스카이워커 Anakin@jedai.com  14 콰이곤 진  15 아미달라 여왕  16 아나킨 어머니  17 자자빙크스(목소리) jaja@jedai.com  18 다스 몰  19 장고 펫  20 마스터 윈두 windu@jedai.com  21 듀크 백작 dooku@jedai.com  <<정답>> --------------------------------------------------------------------------------------------------  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 1, '루크 스카이워커','luke@jedai.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 2, '한 솔로', 'solo@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 3, '레이아 공주', 'leia@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 4, '오비완 케노비', 'Obi-Wan@jedai.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 5, '다쓰 베이더', 'vader@sith.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 6, '다쓰 베이더(목소리)', 'vader-voice@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 7, 'C-3PO', 'c3po@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 8, 'R2-D2', 'r2d2@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 9, '츄바카', 'Chewbacca@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 10, '랜도 칼리시안', 'Chewbacca@alliance.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 11, '요다(목소리)', 'yoda@jedai.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 12, '다스 시디어스', NULL);  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 13, '아나킨 스카이워커', 'Anakin@jedai.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 14, '콰이곤 진', NULL);  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 15, '아미달라 여왕', NULL);  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 16, '아나킨 어머니', NULL);  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 17, '자자빙크스(목소리)', 'jaja@jedai.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 18, '다스 몰', NULL);  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 19, '장고 펫', NULL);  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 20, '마스터 윈두', 'windu@jedai.com');  INSERT INTO CHARACTERS ( CHARACTER\_ID, CHARACTER\_NAME, EMAIL)  VALUES ( 21, '듀크 백작', 'dooku@jedai.com');  COMMIT;  -----------------------------------------------------------------------------------------------------------  6. ROLES 테이블에 다음의 데이터를 저장해보자  ROLE\_ID ROLE\_NAME  ------------------- ------------------------------  1001 제다이  1002 시스  1003 반란군  <<정답>> --------------------------------------------------------------------------------------------------  -- ROLES 테이블 생성  CREATE TABLE ROLES (  ROLE\_ID NUMBER(5,0) NOT NULL,  ROLE\_NAME VARCHAR2(30 BYTE),  CONSTRAINT ROLES\_PK PRIMARY KEY (ROLE\_ID));  -- DATA INSERT  INSERT INTO ROLES ( ROLE\_ID, ROLE\_NAME ) VALUES (1001, '제다이');  INSERT INTO ROLES ( ROLE\_ID, ROLE\_NAME ) VALUES (1002, '시스');  INSERT INTO ROLES ( ROLE\_ID, ROLE\_NAME ) VALUES (1003, '반란군');  COMMIT;  -----------------------------------------------------------------------------------------------------------  7. CHARACTERS 테이블에는 ROLE\_ID 란 컬럼이 있는데, 이 값은 ROLES 테이블의 ROLE\_ID 값을 참조한다.  CHARACTERS 테이블을 변경하여 ROLE\_ID 컬럼이 ROLES 테이블의 ROLE\_ID 값을 참조하도록 참조 키를 생성해보자.  <<정답>> --------------------------------------------------------------------------------------------------  ALTER TABLE CHARACTERS  ADD CONSTRAINT CHARACTERS\_FK FOREIGN KEY ( ROLE\_ID )  REFERENCES ROLES ( ROLE\_ID );  -----------------------------------------------------------------------------------------------------------  8. 참조 키를 생성했으면, 다음으로 CHARACTERS 테이블의 ROLE\_ID 값을 변경해보자.  값의 참조는 ROLES 테이블의 ROLE\_ID 값을 참조한다. 예를 들면 루크 스카이워커, 오비완 캐노비, 요다 등은 제다이 기사이므로 1001 값을 가질 것이며,  다쓰 베이더, 다쓰 몰은 시스 족이므로 1002에 해당한다. 자신이 아는 범위 내에서 이 값을 갱신하는 UPDATE 문장을 작성해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  UPDATE CHARACTERS  SET ROLE\_ID = 1001  WHERE CHARACTER\_ID IN ( 1, 4, 11, 13, 14, 20, 21);    UPDATE CHARACTERS  SET ROLE\_ID = 1002  WHERE CHARACTER\_ID IN ( 5, 6, 12, 18 );    UPDATE CHARACTERS  SET ROLE\_ID = 1003  WHERE CHARACTER\_ID IN (2, 3, 7, 8, 9);    COMMIT;  -----------------------------------------------------------------------------------------------------------  9. CHARACTERS 테이블에는 MASTER\_ID 란 컬럼이 있는데, 이 컬럼의 용도는 EMPLOYEES 테이블의 MANAGER\_ID 와 그 역할이 같다.  즉 제다이나 시스에 속하는 인물 중 그 마스터의 CHARACTER\_ID 값을 찾아 이 컬럼에 갱신하는 문장을 작성해 보자.  제자 마스터  ------------------ -------------------------  아나킨 스카이워커 오비완 캐노비  루크 스카이워크 오비완 캐노비  마스터 윈두 요다  듀크 백작 요다  다쓰 베이더 다쓰 시디어스  다쓰 몰 다쓰 시디어스  오비완 캐노비 콰이곤 진  콰이곤 진 듀크 백작  <<정답>> --------------------------------------------------------------------------------------------------  UPDATE CHARACTERS  SET MASTER\_ID = 4  WHERE CHARACTER\_ID = 13;    UPDATE CHARACTERS  SET MASTER\_ID = 4  WHERE CHARACTER\_ID = 1;    UPDATE CHARACTERS  SET MASTER\_ID = 11  WHERE CHARACTER\_ID = 20;    UPDATE CHARACTERS  SET MASTER\_ID = 11  WHERE CHARACTER\_ID = 21;    UPDATE CHARACTERS  SET MASTER\_ID = 12  WHERE CHARACTER\_ID = 5;  UPDATE CHARACTERS  SET MASTER\_ID = 12  WHERE CHARACTER\_ID = 18;    UPDATE CHARACTERS  SET MASTER\_ID = 14  WHERE CHARACTER\_ID = 4;    UPDATE CHARACTERS  SET MASTER\_ID = 21  WHERE CHARACTER\_ID = 14;  COMMIT;  -----------------------------------------------------------------------------------------------------------  10. CASTING 테이블의 기본 키는 EPISODE\_ID와 CHARACTER\_ID 이다. 이 두 컬럼은 각각 STAR\_WARS와 CHARACTERS 테이블의 기본 키를 참조하고 있다.  CASTING 테이블에 각각 이 두 테이블의 컬럼을 참조하도록 참조 키를 생성해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  -- STAR\_WARS  ALTER TABLE CASTING  ADD CONSTRAINT CASTING\_FK1 FOREIGN KEY ( EPISODE\_ID )  REFERENCES STAR\_WARS ( EPISODE\_ID );    -- CHARACTERS  ALTER TABLE CASTING  ADD CONSTRAINT CASTING\_FK2 FOREIGN KEY ( CHARACTER\_ID )  REFERENCES CHARACTERS ( CHARACTER\_ID );        -----------------------------------------------------------------------------------------------------------  11. 다음 문장을 실행해 보자.  DELETE ROLES  WHERE ROLE\_ID = 1001;  이 문장을 실행하면 그 결과는 어떻게 되는가? 또한 그러한 결과가 나오는 이유는 무엇인지 설명해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  7 번 문제에서 CHARACTERS 테이블에서 참조 키(foreign key)를 생성했기 때문에 삭제할 수 없다.  만약 삭제를 시도하면 무결성 제약조건(HR.CHARACTERS\_FK)을 위배했다는 오류가 난다.  이러한 현상에도 불구하고 삭제를 꼭 해야 한다면, CHARACTERS 테이블에 만들어진 foreign key를 삭제하거나  임시로 foreign key의 상태를 disable 로 한 뒤, 삭제할 수 있다.  (1) foreign key 삭제  ALTER TABLE CHARACTERS  DROP CONSTRAINT CHARACTERS\_FK;    (2) foreign key 의 상태를 disable 상태로...  ALTER TABLE CHARACTERS  MODIFY CONSTRAINT CHARACTERS\_FK DISABLE;  ----------------------------------------------------------------------------------------------------------- |

**2장 인덱스(Index)**

**2.1 인덱스란?**

이 SQL문은 인덱스를 타지 않아서 늦는 거야..

인덱스를 새로 만들었더니 조회 속도가 빨라졌어...

=> 데이터베이스에서의 인덱스도 책에서와 마찬가지로, 테이블에 저장되어 있는 데이터를 좀 더 효율적이고 빠르게 찾기 위해 사용된다.

=> 인덱스를 설정하면 오라클에서 자동으로 인덱스를 검색하여 데이터를 조회한다.

즉, 우리가 할 일은 데이블에 적합한 인덱스를 생성하는 것이고,

모든 작업은 오라클이 자동으로 인덱스를 이용해서 데이터를 검색하고 관리하게 된다.

**2.2 인덱스의 생성**

인덱스는 테이블에 있는 컬럼을 지정하여 만들 수 있다.

제약조건들과는 달리 일반적으로 테이블 생성과 동시에 인덱스를 생성할 수 없다.

- CREATE TABLE문 내에서 생성할 수 없다.

인덱스는 테이블과 같이 별도의 데이터베이스 오브젝트로써 테이블과 동등한 레벨의 객체로서 존

재한다.

따라서 인덱스의 생성은 CREATE문으로 한다.

|  |
| --- |
| **[구문 형식 ]**  **CREATE** [UNIQUE] **INDEX** [스키마명.] 인덱스명  **ON** [스키마명.] 테이블명 ( 컬럼 [,컬럼2, 컬럼3, ...]); |

\*) 인덱스 특징

자신의 스키마에 있을 경우 스키마의 이름 생략 가능

하나의 인덱스는 한 개 이상의 컬럼으로 구성된다.

한 테이블에 여러개의 인덱스를 생성할 수 있다.

\*) 인덱스를 이용한 검색

인덱스가 생성 ==> 오라클은 인덱스 정보를 별도로 저장함

( 여기에는 인덱스 컬럼과 ROWID정보를 담고 있다.)

ex)

EMPLOYEES 테이블의 EMP\_EMAIL\_UK 인덱스를 예로 들면,

1) EMAIL 컬럼에 저장된 값 순서로 정렬한다.

2) 이 값과 ROWID 값을 저장한다.

3) 검색 시 저장된 인덱스 정보를 먼저 찾는다.

4) 인덱스 정보를 통해 실제 테이블에 있는 데이터를 검색하게 된다.

==> 인덱스 정보에는 인덱스 컬럼값과 ROWID 정보가 담겨 있기 때문에 새로운 로우가 삽입

되거나 업데이트, 삭제에 의해 데이터 변경이 일어나면 오라클은 자동으로 저장된 인덱스

정보를 갱신한다.

따라서 한 테이블에 인덱스를 너무 많이 생성하게 되면 SELECT에는 도움이 될 수 있을지 모르

겠지만 데이터가 변경될 때마다 인덱스 정보도 갱신되야 하기 때문에, 전체적으로는 인덱스 관

리에 오라클 자원이 많이 할당되어 성능에 좋지 않은 영향을 주게 된다. 따라서 꼭 필요한 인덱

스만 생성할 필요가 있다.

\*) 단일 인덱스 예 : 한 개의 컬럼에 대해 인덱스 설정

|  |
| --- |
| **CREATE UNIQUE INDEX** "HR"."EMP\_EMAIL\_UK" **ON** "HR"."EMPLOYEES" ("EMAIL");    **CREATE INDEX** "HR"."EMP\_DEPARTMENT\_IX" **ON** "HR"."EMPLOYEES" ("DEPARTMENT\_ID"); |

\*) 복합 인덱스 예 : 두개의 컬럼에 대해 인덱스 설정

생성 가능 컬럼의 개수

1) B-Tree Index : 최대 32

2) BITMAP Index : 최대 30

|  |
| --- |
| **CREATE INDEX** "HR"."EMP\_NAME\_IX" **ON** "HR"."EMPLOYEES" ("LAST\_NAME", "FIRST\_NAME"); |

🡺 UNIQUE Index : 중복값 입력하는 경우 => 무결성 제약조건에 위배됩니다.

중복값을 입력한 뒤 UNIQUE 인덱스를 생성한 경우

=> 중복키가 있습니다. 유일한 인덱스를 작성할 수 없습니다.

\*) UNIQUE키나 기본키를 생성하면 UNIQUE인덱스가 자동으로 생성된다.

UNIQUE인덱스를 생성한다고 해서 UNIQUE나 기본 키 제약조건이 자동으로 생성되지 않는다.

- CREATE INDEX문을 이용하여 사용자가 직접 생성하는 인덱스 ==> 수동 인덱스

- 제약조건에 의해 자동으로 생성되는 인덱스 ==> 자동 인덱스 , 이름은 SYS\_로 시작

\*) 복합 인덱스의 경우 순서가 중요하다.

일반적으로 SELECT 문의 WHERE절에서 사용되는 컬럼 순으로 인덱스를 구성해야 한다.

**2.3 인덱스의 삭제**

|  |
| --- |
| **[구문 형식 ]**  **DROP** **INDEX** [스키마명.] 인덱스명; |

\*) 해당 스키마 사용자로 로그인 했다면(인덱스 소유자라면) 스키마 명은 생략가능

**2.4 인덱스 종류**

- 유일성 여부에 따라

UNIQUE 인덱스 : 인덱스 컬럼에 유일한 값만 입력 가능, 중복값 불허

NON-UNIQUE인덱스 : 중복값 입력 가능

- 인덱스 구성 컬럼의 개수에 따라

단일 인덱스(sigle index) : 인덱스 구성 컬럼 개수가 한 개

복합 인덱스(composite or consatenated index) : 인덱스 구성 컬럼 개수가 두 개 이상

- 인덱스 생성자에 따라

수동 인덱스 : CREATE INDEX 문을 사용하여 사용자가 생성

자동 인덱스 : UNIQUE나 기본 키 생성에 따라 오라클이 자동으로 생성

- 인덱스의 성격/구조에 따라 구분

**B-Tree 인덱스(B-tree index) : 가장 많이 사용되는 인덱스**

**데이터의 저장, 갱신, 삭제가 빈번할 경우 유리**

비트맵 인덱스(bitmap index) : 데량의 데이터에 대한 조회작업이 많을 경우

해쉬 클러스터 인덱스(hash cluster index)

리버스 키 인덱스(Reverse key index)

비트맵 조인 인덱스(Bitmap join index)

함수기반 인덱스(function-based index)

도메인 인덱스(domain index)

분할 인덱스(Partitional index)

**2.5 인덱스 생성시 고려사항**

생성 목적은 테이블에 있는 데이터를 빠르고 저비용으로 검색하기 위함임

- 자주 조회되는 컬럼을 인덱스 컬럼으로 선택한다.

- 참조 제약조건이 있는 컬럼에 생성한다.

- 데이터 량이 많은 테이블에 있어 전체 데이터의 15% 이하의 데이터를 조회할 경우에 인덱스를

생성한다.

15%이상일 경우 테이블 전체를 검색하며 조회하는 것이 빠를 수 있다.

- 테이블 간 조인에 사용된 컬럼을 인덱스 컬럼으로 선택하면 조인 성능이 향상된다.

- 테이블 전체 로우가 적은 경우에는 굳이 인덱스를 만들 필요가 없다.

- 복합 인덱스를 구성할 경우 컬럼의 순서는 SELECT 문의 WHERE 절에서 좀 더 자주 사용되는 컬

럼을 먼저 오게 한다.

- LONG 과 LONG RAW 타입 컬럼은 인덱스로 만들 수 없다.

- 복합 인덱스로 만들 수 있는 최대 컬럼 수는 32개 이다.

- 삽입, 갱신, 삭제가 많이 발생하는 테이블에는 인덱스를 너무 많이 만들지 않도록 한다. 성능에

영향을 준다.

|  |
| --- |
| << 실습문제 >> =============================================================  1. 앞 에서 생성한 CHARACTERS 테이블에는 현재 CHARACTER\_ID 컬럼에 UNIQUE 인덱스가 생성되어 있다.  이번에는 CHARACTER\_NAME 컬럼에 인덱스를 생성하려는데, 이 컬럼에는 어떤 종류의 인덱스를 생성해야 할까?  <<정답>> --------------------------------------------------------------------------------------------------  UNIQUE 인덱스를 생성해도 크게 문제될 것은 없다. 하지만 CHARACTER\_NAME 컬럼은 스타워즈에 출연한 배역의  이름을 저장하고 있는 컬럼이므로, 중복된 데이터가 저장될 가능성이 존재하므로 일반적인 인덱스(NONUNIQUE INDEX)를  생성하도록 한다.  2. CHARACTERS 테이블의 CHARACTER\_NAME에 인덱스를 만들어 보자.  <<정답>> --------------------------------------------------------------------------------------------------  CREATE INDEX CHARACTERX\_IX\_01 ON CHARACTERS ( CHARACTER\_NAME );  ----------------------------------------------------------------------------------------------------------- |

**3장 뷰(View)**

**3.1 뷰란?**

테이블과 매우 흡사한 오브젝트임

뷰와 테이블의 차이점은 뷰는 실제로 데이터를 저장하고 있지 않다.(가상 테이블)

만약 DB 테이블에 주민등록번호 컬럼이 존재하고 다수의 사람에게 정보를 공개하고자 할 때 해

당 컬럼을 제거한 테이블을 제공하고자 한다면?

**3.2 뷰의 생성**

|  |
| --- |
| **-- 일반 쿼리문**  **SELECT** first\_name, last\_name, email, hire\_date  **FROM** employees  **WHERE** department\_id = 20 ; |
| **-- 뷰 생성 쿼리문** **구문형식**  **CREATE** [OR REPLACE ] **VIEW** 스키마 이름, 뷰 이름 AS  **SELECT** 문장 |
| **-- V\_emp1 이라는 이름의 뷰를 생성**  -- 마케팅 부서의 소속 사원 정보  **CREATE VIEW** v\_emp1 **AS**  **SELECT** first\_name, last\_name, email, hire\_date  **FROM** employees  **WHERE** department\_id = 20; |
| **SELECT \* FROM** v\_emp1; |

\*) OR REPLACE : 기존 뷰를 다시 생성하고자 한다면 해당 뷰를 제거하고(DROP VIEW)

다시 생성(CREATE VIEW)해야 한다. 이를 대신하는 문장이다.

\*) 뷰는 쿼리문을 통해서 만들어진다.

일단 뷰가 만들어 지면 테이블과 마찬가지로 접근이 가능하다.

🡺 뷰의 사용 목적은 테이블처럼 사용하면서도 정의된 쿼리의 결과만 볼 수 있으므로 다른 사용자

들에게 데이터의 일부만 공개할 수 있다.

\*) 테이블의 컬럼 정보 확인

|  |
| --- |
| **DESC** v\_emp1;  -- 출력 결과 확인 |

\*) 마약 뷰의 갱신이 필요하다면?

사용자가 마케팅 부서 소속 사원 + 구매부 소속 사원 정보를 필요로 한다면?

[ DROP VIEW ==> CREATE VIEW ] ==[ OR REPLACE ]

|  |
| --- |
| **CREATE OR REPLACE VIEW** v\_emp1 AS  **SELECT** first\_name, last\_name, email, hire\_date  **FROM** employees  **WHERE** department\_id **IN** (20,30);  *CREATE OR REPLACE VIEW Succeeded;*  **SELECT** \* **FROM** v\_emp1; |

\*) 뷰는 쿼리 문장을 통해 생성된다. 따라서 여러 테이블을 조인해서 생성할 수도 있고, 뷰를 사용

하여 새로운 뷰를 생성할 수도 있다.

|  |
| --- |
| -- *EMPLOYEES 테이블을 사용한 뷰만들기*  **CREATE OR REPLACE VIEW** v\_emp\_from\_table **AS**  **SELECT** first\_name, last\_name, email, hire\_date  **FROM EMPLOYEES**  **WHERE** department\_id **IN** (20, 30)  **AND** hire\_date < **TO\_DATE**('1998-01-01'); |
| -- *만들어진 v\_emp1 뷰를 통해 뷰 만들기*  **CREATE OR REPLACE VIEW** v\_emp\_from\_view **AS**  **SELECT** first\_name, last\_name, email, hire\_date  **FROM** v\_emp1  **WHERE** hire\_date < **TO\_DATE**('1998-01-01'); |

**3.3 뷰의 사용**

1) 데이터 보안 : 특정 컬럼만 공개

2) 복잡한 쿼리를 저장하여 사용

3) 뷰를 사용한 데이터 변경

뷰를 통한 데이터 변경은 가능하지만 모든 뷰가 가능한 것은 아니다.

- Read-Only 뷰

- Updatable 뷰 : defalut뷰

=> 뷰의 종류는 생성시점에 결정된다.

|  |
| --- |
| *Read-Only 뷰*  **CREATE [OR REPLACE] VIEW** 스키마이름, 뷰이름 **AS**  **SELECT** 문장  **WITH READ ONLY**; |
| *UPDATABLE 뷰*  **CREATE [OR REPLACE] VIEW** 스키마이름, 뷰이름 **AS**  **SELECT** 문장; |

\*) 앞에서 본 v\_emp1, v\_emp2는 모두 데이터 변경이 가능한 view이다.

ex) UPDATE : ReadOnly View는 데이터 변경 불가

|  |
| --- |
| **CREATE OR REPLACE VIEW** v\_emp1\_read\_only **AS**  **SELECT** first\_name, last\_name, email, hire\_date  **FROM** employees  **WHERE** department\_id **IN** (20,30)  **WITH READ ONLY**;  *CREATE OR REPLACE VIEW succeeded* |
| **CREATE OR REPLACE VIEW** v\_emp1\_update **AS**  **SELECT** first\_name, last\_name, email, hire\_date  **FROM** employees  **WHERE** department\_id **IN** (20,30);  *CREATE OR REPLACE VIEW succeeded* |
| **UPDATE** v\_emp1\_update  **SET** last\_name = '에디슨'  **WHERE** first\_name = '마이클' ;  *1 rows updated* |
| **UPDATE** v\_emp1\_read\_only  **SET** last\_name = '하트쉬타인'  **WHERE** last\_name = '마이클' ;  *SQL Error : ORA-01733 : 가상 열은 사용할 수 없슴니다.* |

ex) INSERT : 실패

|  |
| --- |
| **INSERT INTO** v\_emp1\_update  **VALUES**('길동','홍','HGD', SYSDATE);  *SQL Error : ORA-01400 : NULL을 (“HR”,”EMPLOYEES”.”EMPLOYEE\_ID”) 안에 삽입할 수 없습니다.* |

\*) INSERT는 불가능하다.

기준테이블인 EMPLOYEES의 경우 EMPLOYEES\_ID가 기본 키이며 NOT NULL속성을 가지고

있는데 NULL을 넣어서 오류가 발생한다.

Update를 통해 데이터 변경작업을 수행하면 실제 테이블에 있는 데이터가 변경된다.

DELETE는 가능하다.

ex) 모든 기능을 수행하는 VIEW 생성

|  |
| --- |
| **CREATE OR REPLACE VIEW** v\_emp1\_update AS  **SELECT** employee\_id, first\_name, last\_name, email, hire\_date, job\_id, department\_id  **FROM** employees  **WHERE** department\_id **IN** (20,30);  *CREATE OR REPLACE VIEW succeeded* |
| **INSERT INTO** v\_emp1\_update  **VALUES**(9999, '길동','홍', 'HHK', SYSDATE, 'IT\_PROG', 20);  *1 rows inserted* |
| **DELETE** v\_emp1\_update  **WHERE** employee\_id = 9999;  *1 rows deleted* |

\*) 뷰를 생성할 경우 먼저 만들고자 하는 뷰의 성격을 고려해야 한다.

\*) 뷰나 테이블에서 어떤 컬럼들이 INSERT, UPDATE, DELETE가 가능한지 알 수 있다.

**SELECT** \*

**FROM** USER\_UPDATABLE\_COLUMNS

**WHERE** table\_name = 'V\_EMP1\_UPDATE' ;

**3.4 뷰의 옵션**

WITH CHECK OPTION

뷰 생성시 해당 구문을 첨가하면 뷰를 정의하는 쿼리문의 조건에 어긋난 데이터의 INSERT가 불가

능하다.

테이블 생성시 CHECK 제약조건과 동일한 역할이다.

|  |
| --- |
| *-- INSERT 가능*  **INSERT INTO** v\_emp1\_update  **VALUES**(777, '조영','대', 'DJY', SYSDATE, 'FI\_MGR', 40);  *1rows inserted* |

ex) 옵션 설정 테이블 생성

|  |
| --- |
| **CREATE OR REPLACE VIEW** v\_emp1\_update **AS**  **SELECT** employee\_id, first\_name, last\_name, email, hire\_date, job\_id, department\_id  **FROM** employees  **WHERE** department\_id **IN** (20,30)  **WITH CHECK OPTION**;  *CREATE OR REPLACE VIEW succeeded* |
| **INSERT INTO** V\_EMP1\_UPDATE  **VALUES**(778, '중상','대', 'DJS', SYSDATE, 'FI\_MGR', 40);  *SQL Error : ORA-01402 : 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다.* |

🡺 READ\_ONLY부와 UPDATEABLE뷰는 단 하나의 테이블에서 데이터를 가져옴

\*) 2개의 테이블을 이용한 뷰 생성

|  |
| --- |
| **CREATE OR REPLACE VIEW** v\_emp1\_dual\_update **AS**  **SELECT** a.employee\_id, a.first\_name, a.last\_name, a.email, a.hire\_date, a.job\_id,  a.department\_id emp\_department,  b.department\_id, b.department\_name  **FROM** employees a,  departments b  **WHERE** a.department\_id = b.department\_id  **AND** b.department\_id IN (20,30);  *CREATE OR REPLACE VIEW succeeded* |
| -- 삽입  **INSERT INTO** v\_emp1\_dual\_update  ( employee\_id, first\_name, last\_name, email, hire\_date, job\_id, emp\_department )  **VALUES**(779, '인귀','설', 'SIK', SYSDATE, 'FI\_MGR', '30');  *1 rows inserted* |
| -- 삽입  **INSERT INTO** v\_emp1\_dual\_update (department\_id, department\_name)  **VALUE**(900, '테스트1');  *SQL Error : ORA-01779 : 키-보존된 것이 아닌 테이블로 대응한 열을 수정할 수 없습니다.* |
| *-- 수정*  ***UPDATE*** *v\_emp1\_dual\_update*  ***SET*** *email = 'DaeJoYoung'*  ***WHERE*** *employee\_id = 779;*  *1 rows updated*  ***UPDATE*** *v\_emp1\_dual\_update*  ***SET*** *department\_name = '경영지원부2'*  ***WHERE*** *employee\_id = 779;*  *SQL Error : ORA-01779 : 키 보존된 것이 아닌 테이블로 대응한 열을 수정할 수 없습니다.* |

\*) INSERT : WITH CHECK OPTION을 사용하면 INSERT는 할 수 없고, INSERT 되는 컬럼들은 키-보존

테이블의 컬럼이어야 한다.

키-보존 테이블 🡺 뷰를 구성하는 기준 테이블 중 모든 PRIMARY혹은 UNIQUE키 값이 뷰 자체

에서도 UNIQUE한 값을 가진 테이블을 말함

\*) UPDATE : WITH CHECK OPTION을 사용할 수 없으며 모든 UPDATE되는 테이블은 키-보존 테이블

의 컬럼이어야 한다.

\*) WITH CHECK OPTION을 뺴고 삽입을 수행하면 데이터 입력이 가능하다.

|  |
| --- |
| << 실습문제 4-3 >> =============================================================  1. 이전에 만들었던 CHARACTERS 테이블의 CHARACTER\_ID, CHARACTER\_NAME, EMAIL 컬럼의 데이터를 OE 사용자가 볼 수 있도록 JEDAI\_V란 이름으로 뷰를 생성해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  -- JEDAI\_V 생성  CREATE OR REPLACE VIEW JEDAI\_V AS  SELECT CHARACTER\_ID, CHARACTER\_NAME, EMAIL  FROM CHARACTERS;    --OE 사용자에게 JEDAI\_V의 SELECT 권한을 준다.  GRANT SELECT ON JEDAI\_V TO OE;  -- OE 사용자로 로그인 한 뒤, JEDAI\_V를 조회해본다.(JEDAI\_V가 아닌 HR.JEDAI\_V로 표기해야 조회할 수 있다)  SELECT \*  FROM HR.JEDAI\_V;  -----------------------------------------------------------------------------------------------------------  2. JEDAI\_V 뷰를 사용하여 ‘루크 스카이워커’의 이메일 정보를 ‘luke2@jedai.com’로 변경하는 UPDATE 문을 작성하여 실행해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  --OE 사용자로 다음문장을 실행할 경우, 오류 발생  UPDATE HR.JEDAI\_V  SET email = 'luke2@jedai.com'  WHERE character\_id = 1;    -- HR 사용자로 실행하면 성공  UPDATE JEDAI\_V  SET email = 'luke2@jedai.com'  WHERE character\_id = 1;    -----------------------------------------------------------------------------------------------------------  3. 2 번 문제의 결과는 어떻게 되는가? 또한 왜 그러한 결과가 나오게 되었는지 설명해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  OE 사용자에게는 JEDAI\_V의 SELECT 권한만 있으므로 UPDATE 문을 실행하게 되면 권한이 불충분하다는 오류메시지가 나오게 된다.  따라서 OE 사용자가 2번의 UPDATE 문을 실행하기 위해서는 UPDATE 권한을 주거나, HR 사용자로 로그인하여 UPDATE 문을 실행해야 한다.  (1) HR 사용자로 로그인  -- 다시 HR 사용자로 로그인한 뒤, 다음 문장을 실행한다.  UPDATE JEDAI\_V  SET email = 'luke2@jedai.com'  WHERE character\_id = 1;    COMMIT;  -- JEDAI\_V의 경우 CHARACTERS 테이블의 키 값을 가지므로 키보존 테이블이 되므로 UPDATE 가 가능한 것이다.    (2) OE사용자에게 UPDATE 권한 부여(HR 사용자로 로그인해야 한다)  GRANT UPDATE ON JEDAI\_V TO OE;  UPDATE HR.JEDAI\_V  SET email = 'luke2@jedai.com'  WHERE character\_id = 1    COMMIT; |

**4장 시노님(Synonym) : 객체에 호를 만들자.**

스키마 오브젝트들의 별칭을 말한다.

즉 테이블, 뷰, 시퀀스, 프로시저, 함수, 패키지, 타입, 사용자 정의타입 다른 시노님들의 가명을 지

칭한다

시노님이 가명 혹은 별칭을 의미하므로 뷰와 마찬가지로 실제로는 데이터를 저장하고 있지 않으며

시노님의 정의만 데이터 딕셔너리에 저장되어 있다.

ex) OE 사용자는 원칙적으로 HR 스키마의 EMPLOYEES 테이블의 데이터를 조회할 수 없다.

만약 조회하고 싶다면 HR 혹은 SYS 사용자가 EMPLOYEES 테이블의 SELECT 권한을 부여해야

만 가능하다.

OE 사용자가 권한을 부여 받았다고 가정하고, EMPLOYEES 테이블을 조회해보자.

SELECT \* FROM employees;

==> 오류 발생 : 테이블 또는 뷰가 존재하지 않습니다.

why?) 권한은 있지만 스키마가 서로 다르다.

EMPLOYEES 테이블의 정식 명칭은 HR.EMPLOYESS 이다.

HR사용자가 로그인시는 스키마명이 필요없지만,

OE 사용자로 로그인시는 반드시 스키마명을 명시해야 한다.

만약, HR.EMPLOYEES의 가명을 EMPLOYEES 라고 만들어 버리면

OE 입장에서 편하게 테이블명으로 접근할 수 있다.

또한 이렇게 되면 EMPLOYEES 라는 객체가 동일한 이름으로 2개 존재하지만 스키마가

다르므로 충돌되지도 않고 EMPLOYEES 가 HR 스키마에 속한다는 사실을 감출 수 있으므

로 데이터 보안성 측면에서도 시노님을 사용하는 것의 장점이 있다.

- OE.EMPLOYEES : OE 스키마의 EMPLOYEES 라는 시노님

- HR.EMPLOYEES : HR 스키마의 EMPLOYEES 라는 테이블

**4.1 시노님 종류**

PUBLIC 시노님 : 공개용 => 모든 사용자가 이 시노님에 접근 가능

PRIVATE 시노님 : 생성한 스키마에서만 접근 가능

🡺 시노님 사용 제약 사항

1. HR.EMPLOYEES 테이블에 대한 SELECT 권한이 있어야 한다. SELECT 권한이 없더라도 시노님 생성은 가능하지만 데이터는 조회할 수 없다.
2. 시노님 생성 권한이 있어야 한다.

- public 시노님 생성 : CREATE PUBLIC SYNONYM 권한이 있어야 한다.

- private 시노님 생성 : CREATE SYNONYM(자신의 스키마에서 생성할 경우)

CREATE ANY SYNONYM(다른 스키마에서 생성) 권한이 있어야 함

**4.2 시노님 생성**

|  |
| --- |
| [ 구문 형식 ]  **CREATE [OR REPLACE ] [PUBLIC ] SYNONYM** '[스키마명].시노님명'  **FOR** '스키마명.대상 오브젝트명'; |
| [ ex ]  **CREATE OR REPLACE PUBLIC** **SYNONYM** pub\_emp  **FOR** hr.employees; |

\*) 디폴트값 : PRIVATE

PUBLIC 지정시 스키마명 생략 => 공개용이기 때문

\*) 시노님 생성시 소유자는?

PUBLIC : PUBLIC : 실 사용자가 아니라 룰(Role)이다.

PRIVATE : 생성 스키마

\*) 대표적인 PUBLIC 시노님 ==> DUAL

DUAL은 원래 SYS 스키마에 속한 테이블이다.

SYS.DUAL 테이블을 PUBLIC 시노님으로 생성했기 때문이다.

**4.3 시노님 제거**

|  |
| --- |
| [ 구문 형식 ]  **DROP SYNONYM** '스키마명.시노님명' |
| [ 구문 형식 ]  **DROP PUBLIC SYNONYM** '시노님명'; |

|  |
| --- |
| << 실습문제 4-4 >> =============================================================  1. 앞 절에서 생성한 JEDAI\_V 뷰에 대해 모든 사용자가 JEDAI\_CHARS 란 객체명으로 해당 데이터를 조회할 수 있도록 JEDAI\_CHARS 라는 이름으로  시노님을 만들어 보자.  <<정답>> --------------------------------------------------------------------------------------------------  -- 모든 사용자가 참조가능해야 하므로 PUBLIC 시노님을 생성한다. (HR사용자로 로그인 한다)  CREATE OR REPLACE PUBLIC SYNONYM JEDAI\_CHARS FOR HR.JEDAI\_V;  -----------------------------------------------------------------------------------------------------------  2. JEDAI\_V 뷰에 대해 OE 사용자만 참조 가능하도록 MY\_JEDAI\_CHARS 란 이름의 시노님을 생성해 보자.  <<정답>> --------------------------------------------------------------------------------------------------  CREATE OR REPLACE SYNONYM OE.MY\_JEDAI\_CHARS FOR HR.JEDAI\_V;  ----------------------------------------------------------------------------------------------------------- |

**5장 시퀀스(Sequence)**

-- 시퀀스( 순번을 부여하자.)

어떤 테이블의 기본 키가 숫자형 데이터 타입이다.

해당 키는 특별한 조건이 없고 각각 입력되는 로우를 식별하기 위함이다.

일반적으로 이 기본 키 값에 시퀀스에 의해 생성된 값을 할당하게 된다.

\*) 시퀀스 생성

|  |
| --- |
| **CREATE SEQUENCE** "HR"."EMPLOYEES\_SEQ"  MINVALUE 1  MAXVALUE 999999999999999999999999999  INCREMENT BY 1  START WITH 207  NOCACHE  NOORDER  NOCYCLE ; |

=> MINVALUE : 시퀀스가 시작되는 최초의 숫자

MAXVALUE : 시퀀스가 끝나는 최대 숫자

INCREMENT : 증가되는 단위로 28자리 숫자까지 올 수 있다. 0은 올 수 없다.

양수값 음수값이 모두 올 수 있다.

START WITH : 시퀀스 생성이 시작되는 값을 명시

생략시 오름/내림에 따라 MINVALUE/MAXVALUE가 된다.

CACHE|NOCACHE : 오라클에서 시퀀스를 생성하기 위해 미리 값을 할당해 놓기 때문에 시퀀스에

좀 더 빠르게 접근이 가능하다.

동시 사용자가 많을 경우 이 옵션을 선택한다.

NOCACHE : 미리 값을 할당하지 않는다.

ORDER|NOORDER : ORDER를 사용하면 요청되는 순서대로 값을 생성한다.

디폴트값은 NOORDER이다.

CYCLE|NOCYCLE : 생성된 시퀀스 값이 최대치 혹은 최소치에 다다랐을 경우 초기값부터 다시 시

작할지 여부 디폴트값은 nocycle이다.

\*) 시퀀스의 삭제는 DROP문을 사용한다.

DROP SEQUENCE "스키마명.시퀀스명";

|  |
| --- |
| << 실습문제 4-5 >> =============================================================  1. CREATE AS 구문을 사용해 CHARACTERS 테이블을 복사해서 CHARS2 란 이름으로 테이블을 생성하라.  <<정답>> --------------------------------------------------------------------------------------------------  CREATE TABLE CHARS2 AS  SELECT \*  FROM CHARACTERS;  -----------------------------------------------------------------------------------------------------------  2. CHARS2 테이블에 있는 모든 데이터를 삭제한 뒤, CHARS\_SEQ란 이름으로 초기값은 1로 시작하고 1씩 증가하도록 시퀀스를 생성하라.  <<정답>> --------------------------------------------------------------------------------------------------  DELETE CHARS2;  COMMIT;  CREATE SEQUENCE CHARS\_SEQ  MINVALUE 1  MAXVALUE 999999999999999999999999999  INCREMENT BY 1  START WITH 1  NOCACHE  NOORDER  NOCYCLE;  -----------------------------------------------------------------------------------------------------------  3. CHARS\_SEQ 시퀀스를 사용하여 CHARS2 테이블에 CHARACTERS 테이블과 동일한 내용의 데이터를 입력해보자.  <<정답>> --------------------------------------------------------------------------------------------------  INSERT INTO chars2 ( character\_id, character\_name, master\_id, role\_id, email )  SELECT chars\_seq.nextval, character\_name, master\_id, role\_id, email  FROM characters;  COMMIT;  -----------------------------------------------------------------------------------------------------------    -- CASTING TABLE DATA INSERT  INSERT INTO casting VALUES ( 4, 1, '마크 해밀');  INSERT INTO casting VALUES ( 4, 2, '해리슨 포드');  INSERT INTO casting VALUES ( 4, 3, '캐리 피셔');  INSERT INTO casting VALUES ( 4, 4, '알렉 기네스');  INSERT INTO casting VALUES ( 4, 5, '데이비드 프로우즈');  INSERT INTO casting VALUES ( 4, 6, '제임스 얼 존스');  INSERT INTO casting VALUES ( 4, 7, '안소니 다니엘스');  INSERT INTO casting VALUES ( 4, 8, '케니 베이커');  INSERT INTO casting VALUES ( 4, 9, '피터 메이휴');  INSERT INTO casting VALUES ( 5, 1, '마크 해밀');  INSERT INTO casting VALUES ( 5, 2, '해리슨 포드');  INSERT INTO casting VALUES ( 5, 3, '캐리 피셔');  INSERT INTO casting VALUES ( 5, 4, '알렉 기네스');  INSERT INTO casting VALUES ( 5, 5, '데이비드 프로우즈');  INSERT INTO casting VALUES ( 5, 6, '제임스 얼 존스');  INSERT INTO casting VALUES ( 5, 7, '안소니 다니엘스');  INSERT INTO casting VALUES ( 5, 8, '케니 베이커');  INSERT INTO casting VALUES ( 5, 9, '피터 메이휴');  INSERT INTO casting VALUES ( 5, 10, '빌리 디 윌리엄스');  INSERT INTO casting VALUES ( 5, 11, '프랭크 오즈');  INSERT INTO casting VALUES ( 6, 1, '마크 해밀');  INSERT INTO casting VALUES ( 6, 2, '해리슨 포드');  INSERT INTO casting VALUES ( 6, 3, '캐리 피셔');  INSERT INTO casting VALUES ( 6, 4, '알렉 기네스');  INSERT INTO casting VALUES ( 6, 5, '데이비드 프로우즈');  INSERT INTO casting VALUES ( 6, 6, '제임스 얼 존스');  INSERT INTO casting VALUES ( 6, 7, '안소니 다니엘스');  INSERT INTO casting VALUES ( 6, 8, '케니 베이커');  INSERT INTO casting VALUES ( 6, 9, '피터 메이휴');  INSERT INTO casting VALUES ( 6, 10, '빌리 디 윌리엄스');  INSERT INTO casting VALUES ( 6, 11, '프랭크 오즈');  INSERT INTO casting VALUES ( 6, 12, '이언 맥디어미드');  INSERT INTO casting VALUES ( 1, 13, '제이크 로이드');  INSERT INTO casting VALUES ( 1, 4, '이완 맥그리거');  INSERT INTO casting VALUES ( 1, 14, '리암 니슨');  INSERT INTO casting VALUES ( 1, 15, '나탈리 포트만');  INSERT INTO casting VALUES ( 1, 11, '프랭크 오즈');  INSERT INTO casting VALUES ( 1, 16, '페닐라 어거스트');  INSERT INTO casting VALUES ( 1, 12, '이언 맥디어미드');  INSERT INTO casting VALUES ( 1, 7, '안소니 다니엘스');  INSERT INTO casting VALUES ( 1, 8, '케니 베이커');  INSERT INTO casting VALUES ( 1, 17, '아흐메드 베스트');  INSERT INTO casting VALUES ( 1, 18, '레이 파크');  INSERT INTO casting VALUES ( 2, 4, '이완 맥그리거');  INSERT INTO casting VALUES ( 2, 15, '나탈리 포트만');  INSERT INTO casting VALUES ( 2, 13, '헤이든 크리스텐슨');  INSERT INTO casting VALUES ( 2, 11, '프랭크 오즈');  INSERT INTO casting VALUES ( 2, 12, '이언 맥디어미드');  INSERT INTO casting VALUES ( 2, 16, '페닐라 어거스트');  INSERT INTO casting VALUES ( 2, 17, '아흐메드 베스트');  INSERT INTO casting VALUES ( 2, 7, '안소니 다니엘스');  INSERT INTO casting VALUES ( 2, 8, '케니 베이커');  INSERT INTO casting VALUES ( 2, 19, '테무엘라 모리슨');  INSERT INTO casting VALUES ( 2, 20, '사무엘 L. 잭슨');  INSERT INTO casting VALUES ( 2, 21, '크리스토퍼 리');  INSERT INTO casting VALUES ( 3, 4, '이완 맥그리거');  INSERT INTO casting VALUES ( 3, 15, '나탈리 포트만');  INSERT INTO casting VALUES ( 3, 13, '헤이든 크리스텐슨');  INSERT INTO casting VALUES ( 3, 11, '프랭크 오즈');  INSERT INTO casting VALUES ( 3, 12, '이언 맥디어미드');  INSERT INTO casting VALUES ( 3, 17, '아흐메드 베스트');  INSERT INTO casting VALUES ( 3, 7, '안소니 다니엘스');  INSERT INTO casting VALUES ( 3, 8, '케니 베이커');  INSERT INTO casting VALUES ( 3, 19, '테무엘라 모리슨');  INSERT INTO casting VALUES ( 3, 20, '사무엘 L. 잭슨');  INSERT INTO casting VALUES ( 3, 21, '크리스토퍼 리');  COMMIT; |